(2)

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

**DESIGN AND IMPLEMENTATION
OF A MULTIMEDIA DBMS :
MODIFICATION AND DELETION**

by

Rosemary Ellen Stewart

September 1991

Thesis Advisor: Vincent Y. Lum

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Computer Science Dept. Naval Postgraduate School 6b. OFFICE SYMBOL (if applicable) CS37	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
DESIGN AND IMPLEMENTATION OF A MULTIMEDIA DBMS: MODIFICATION AND DELETION (U)

12. PERSONAL AUTHOR(S)
STEWART, Rosemary Ellen

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Master's Thesis	FROM 08/89 TO 09/91	September 1991	244

16. SUPPLEMENTARY NOTA The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Multimedia Database Management System, Multimedia, DBMS, MDBMS, Media Database: Retrieval, Deletion and Modification.

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

At the Naval Postgraduate School, computer science students are currently working on a multimedia database management (MDBMS) project. This prototype designed in 1988, has the ability to capture, store, manage, retrieve and present both standard data, like alphanumerics and numerics, and media data. Media data in this thesis refers to graphics, signals, sound and image and is stored using the abstract data type (ADT) concept. The MDBMS is built upon a conventional INGRES DBMS using ADT's. The multimedia database management system (MDBMS) can integrate audio, image, and formatted data so that these forms of data can process in the following ways: create tables, insert, delete, and retrieve. A complete database management system, requires deletion and modification operations to remove data already stored in a MDBMS or modify data in MDBMS storage. Formatted data is passed directly to INGRES for all types of processing. However, the inclusion of media data types in the MDBMS requires additional data structures and applications for modifying data that the INGRES catalog management cannot process directly. The special handling, SQL operations required to process data are discussed. This thesis concentrates on the design and implementation operations for deletion and modification of formatted and unformatted data in the MDBMS.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Vincent Y. Lum	22b. TELEPHONE (Include Area Code) (408) 646-2175 / 22c. OFFICE SYMBOL CSLm

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Approved for public release; distribution is unlimited

DESIGN AND IMPLEMENTATION OF A MULTIMEDIA DBMS: MODIFCATION AND DELETION

by
Rosemary Ellen Stewart
Captain, United States Army
B.S., United States Military Academy, 1982

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1991

Author: ______________________
Rosemary Ellen Stewart

Approved By: ______________________
Vincent Y. Lum, Thesis Advisor

____
C. Thomas Wu, Second Reader

____
Robert B. McGhee, Chairman,
Department of Computer Science

ABSTRACT

At the Naval Postgraduate School, computer science students are currently working on a multimedia database management (MDBMS) project. This prototype designed in 1988, has the ability to capture, store, manage, retrieve and present both standard data, like alphanumerics and numerics, and media data. Media data in this thesis refers to graphics, signals, sound and image and is stored using the abstract data type (ADT) concept. The MDBMS is built upon a conventional INGRES DBMS using ADT's. The multimedia database management system (MDBMS) can integrate audio, image, and formatted data so that these forms of data can process in the following ways: create tables, insert, delete, and retrieve. A complete database management system, requires deletion and modification operations to remove data already stored in a MDBMS or modify data in MDBMS storage. Formatted data is passed directly to INGRES for all types of processing. However, the inclusion of media data types in the MDBMS requires additional data structures and applications for modifying data that the INGRES catalog management cannot process directly. The special handling, SQL operations required to process data are discussed. This thesis concentrates on the design and implementation operations for deletion and modification of formatted and unformatted data in the MDBMS.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENT

I am grateful to all the instructors and students for their interest and research in the multimedia database management (MDBMS) project who, at this point, are too numerous to mention by name but are referenced heavily in my thesis. Without their work, I would not have become involved with the prototype. I thank Daniel Keim for his contributions and valuable input to the MDBMS prototype, and my thesis companions, Charles A. Peabody and Huseyin Aygun, with whom I worked as a team to develop the current MDBMS. I express sincere thanks to Professor Vincent Lum, Daniel Keim and Suleyman Bayramoglu for their encouragement and support in the conception and preparation of writing my thesis.

I. INTRODUCTION

A. BACKGROUND

The increasing demands in automating many applications intensifies the need for ways to gather, store and manage varied forms of data. Today's computer technology allows better and more efficient methods to perform many operations. Standard data, in the form of numerics and alphanumerics, is easy to handle and requires relatively small storage space in database management systems (DBMS). However, other types of data are needed in the workplace today. The medical, educational, and military fields are examples of areas that need instantaneous access to graphics, photos, images, sounds, signals and videos in conjunction with standard text data. These forms of data are generally referred to as media data or sometimes as unformatted data. Multimedia database management systems (MDBMS) handle the storage, retrieval, and manipulation of media (or unformatted) as well as standard (or formatted) data.

A major operation on accessing DBMS data is by means of content search. In a MDBMS, content search is difficult to achieve since the media data is unstructured, complex, and intrinsically rich in semantics. One way to build on the content search methodology and to simplify the work involved is to perform the content search on verbal descriptions of the multimedia data. A project using this technique is introduced here.

In the Computer Science Department at the Naval Postgraduate School, a multimedia database project was established in 1988. The goa! of the project is to build a prototype multimedia database management system (MDBMS) for capturing, storing, managing, retrieving and presenting both standard and media data. This project, called the Multimedia Database Management System (MDBMS) Project, integrates audio, image and formatted data providing the ability to create tables for the storage of data and to insert, delete, and retrieve the data. [Ref. 9]

Each media graphics, image, signal or sound type has peculiarities in capture, storage and retrieval because of the varied manner that the media type is comprised. Sound media data requires sampling rate, frequency and duration. Image data requires pixel size information. The space to store these objects in a computer varies from a few thousand bytes to several megabytes. In the MDBMS prototype these media data types are represented as one unique value or object using the Abstract Data Type (ADT) concept. This concept allows each media type to be treated as one unique value of the type "media" [Ref. 10]. The ADT concept permits the media data to be handled by the conventional database management operations of create, retrieve, modify and delete.

B. RELATED WORK

In other locations around the globe, research projects involved in the processing of multimedia data are devising ways to manage complex media data. At the University of Waterloo, a team developed the MINOS project. This system manages highly structured multimedia objects that consist of attributes as well as text, image, and voice parts. MINOS is an object-oriented multimedia information system that provides integrated facilities for creating and managing complex multimedia objects. Intricate browsing and user interface features facilitate the browsing of the schema and synchronized updates. [Ref. 5] A database program that undertook several multimedia projects by establishing database requirements for multimedia applications is the MCC Database Program, ORION [Ref. 21]. In this program, requirements for a data model and for the sharing and manipulation of multimedia data were identified. ORION is an object-oriented database management system developed at MCC in Austin, Texas. ORION contains a Multimedia Information Manager (MIM) for processing multimedia data [Ref. 21]. MODES1 and MODES2 are two "mixed-object database systems" developed in the IBM Tokyo Research Laboratory [Ref. 8]. An ESPRIT project designing a multimedia filing system called MULTOS [Ref. 2, Ref. 3] is a European contribution to the field. These projects are discussed in detail in [Ref. 10, Ref. 11, Ref. 12] and will not be presented in more depth in this thesis.

Other related research areas are hypertext and hypermedia for individual computer systems. This concept of hypertext came into the computer systems arena in the 1960's.

Initially designed to manage linked text segments, it now has become "hypermedia" in its extended version of managing images and sound [Ref. 10, Ref. 11]. The ARGOS project being developed at the Naval Postgraduate School uses Macintosh computers with a hypercard application [Ref. 21]. Hypertext and hypermedia data management uses the hierarchical data structure technique that forces queries to follow a hierarchical tree structure to process media data. This method does not permit queries of the data as in conventional database management systems. An interpreter is required to process the user commands. Because hypertext and hypermedia are found on personal computer systems and are not networked, the sharing of data as in most database systems does not usually occur. The MDBMS was initially designed to overcome the shortcomings and disadvantages of hypertext and hypermedia systems.[Ref. 21]

C. SCOPE OF THE THESIS

The MDBMS project continues to conduct research and improve its MDBMS prototype. The design of the system and many of the operations, including table creation, data insertion, and data retrieval, have been presented by Meyer-Wegener [Ref. 11], Lum [Ref. 10], Atila [Ref. 1], Pongsuwan [Ref. 15], and Pei [Ref. 14]. Thesis by Aygun, [Ref. 2] covers complex retrieval and nested query operations for the MDBMS. A natural language parser in PROLOG was created that would understand the meaning of the natural language captions which describe the content of the media data. The parser recognizes the syntax and semantics of the natural language descriptions the user makes in a query. The parser interacts with the multimedia database to locate the appropriate data items [Ref. 6, Ref. 7, Ref. 16]. The parser, designed for content based search for media data, aids the user in the retrieval of the image description stored with the actual image.

The current version of the prototype has the ability to create the tables, input both formatted and unformatted data, and to retrieve that data from the database. The MDBMS needs deletion and modification portions to provide all the basic database management operations. This thesis will concentrate on the complex methods and operations of data modification and deletion in the MDBMS specifically the deletion and modification of data involving image and sound and the integration of media with formatted data. With the completion of the deletion and modification procedures of the NPS MDBMS project, the prototype will be a database management system complete with the capability to manage different types of media data.

D. OVERVIEW OF THESIS

This thesis comprises six chapters and four appendices. Chapter II provides the discussion of the previous work done and the background development of the MDBMS project. The chapter covers key features of the system environment, including the software and hardware architecture of the system, and the data structure design for the implementation of the system modules. The design for the deletion and modification sequences are first mentioned in this chapter. Chapter III, presents the modularization work on the MDBMS project programming code. Chapter IV concentrates on the design and implementation of how information already stored in the database is deleted. This deletion procedure implementation is an important preliminary step to replacing and modifying existing code in a MDBMS. Chapter V concentrates on the details of the design and the implementation operations of the modification procedures. Chapter VI, summarizes the work and its performance by presenting conclusions and recommendations for future work, as well as work currently in progress or planned. The program code for the modification and deletion modules appear in the Appendices. Appendix A presents the deletion procedures sequence code as arranged in the Retrieve.c module. Appendix B has the modification procedures code. Appendix C has an example of running the database. Appendix D consists of the Makefile and sample parser dictionary.

II. SURVEY OF MDBMS

A multimedia database management system (MDBMS) is defined as a system that manages media data: image, text, voice, and signals along with formatted data. Several mechanisms are necessary in the MDBMS to manage data properly; e.g. concurrency control, consistency checkers, and recovery capabilities. Provisions for query language and query processing are also requirements for a MDBMS to operate. The MDBMS architecture is given in this chapter, including the actual hardware, software and interconnections of the MDBMS. All areas have evolutionary discussions explaining their current state of development. The data organization for multimedia objects and the manner in which the media objects interface with the conventional data are important concepts to be explained. Presentation of the natural language parser and the implementation of the running modules of the MDBMS are also presented in this chapter. Previous work on the MDBMS is discussed in detail in the works of Atila [Ref. 1], Pei [Ref. 14], and Pongsuwan [Ref. 15]. Enough information to familiarize the reader with the general hardware and software environment and the assumptions made for the research are given without restating the previous theses.

A. DEVELOPMENT OF MDBMS

The initial 1988 hardware architecture for the MDBMS proposed by Lum [Ref. 9], was changed over time as more individuals worked on the project. The major concern of all involved was to have an architecture that would allow the processing of multimedia data as timely and conveniently as it is to process standard formatted data within a conventional database. The MDBMS is a system developed on top of an existing DBMS such that it allows the processing of media data. The Intelligent Retrieval Subsystem, the natural language parser and matcher, are added to define operations that would help support content search. Content search is important in order to allow an easy but powerful retrieval of the multimedia data. Content search is difficult to achieve because of the complexity and the different characteristics inherent to the different media types. A content search, performed on the media attributes only, is the task of locating the media data corresponding to a specified content. An example of a content search is a database with ships. How can we determine what type of ship is pictured in the photo? To find out if the ship is a cruiser, destroyer, or submarine, more information aside from the photo is necessary. Another example is an image of a weapon. How do we know if it is one carried on a ship or one carried by a sailor? Or if the weapon is a U. S. weapon or from another country? A very observant individual needs to analyze the photos and record all the contents in such a manner that the contents could be retrieved based on objects or actions seen in the photo. Technology today is not able to answer the question on the type of ship or the make of the weapon. We can locate this information using two areas of research: Artificial Intelligence(AI) and Information Retrieval (IR). We can define the contents of the photo (multimedia data) into text captions and use the text description equivalent to the media data to match and then retrieve the data.

B. SYSTEM ENVIRONMENT

The MDBMS prototype consists of hardware, software, and the interaction of these two configurations. In this section, the hardware, the software, and the implementation of prototype are presented. In the discussion of the implementation, the internal structures necessary for the module operations are mentioned.

1. Hardware Configuration

The hardware design of the prototype MDBMS is a SUN-3 workstation in a UNIX environment connected by an ETHERNET to an IBM PC. The IBM PC is used to manage the sound data. The hardware design requires the use of the multiple languages and management systems to make the system work.

2. Software Configuration

As mentioned, the MDBMS prototype was built on top of an existing DBMS system, INGRES, to support formatted and multimedia data. INGRES has the ability to manage stored data. INGRES possesses many positive traits desired in a DBMS. However, INGRES does have many restrictions which prevent it from being the optimal management system such as it does not support the ADT concept. The ADT concept provides the best way to perform data storage for the data management of media data, and, therefore, it was chosen to implement the MDBMS prototype. INGRES uses an internal catalog system. This type of catalog system prevents the user from readily accessing data already stored. INGRES does support embedded SQL with C as a host language. The user must pre-compile the SQL statements into INGRES low level code for execution where no high level function calls are available. Query statements to the database are in the embedded SQL (ESQLC) format. [Ref. 15] Examples of the SQL statements are found in Chapter IV and V.

C is the main language used in this prototype system. This language is a general purpose programming language and is generally used in the coding to build user utilities and interfaces for this project and other systems. INGRES is used as the information base since it is flexible enough to manage the unformatted data information within its catalog management facility. Early on in the project, a version of the INGRES catalog management was chosen to organize the data tables. Newer versions of INGRES have since been available on the commercial market and provide better, more efficient operations for the MDBMS than the one being used. However, the budgeted cost for this prototype, which is only a demonstration model, did not justify the enormous expenditures for the newer INGRES version. The recoding of all the prototype code completed thus far on the database project was another reason that no conversion to the newest software was made. In addition, the IBM PC is necessary to maintain the audio portion of the database because management of sound was not available in the early SUN workstations.[Ref. 1, Ref. 15]

ETHERNET, a Local Area Network (LAN), is the mode in which the sound data is transferred to the SUN Workstation from the IBM PC. This transport system software provides the conduit that allows the sound data to pass through to the rest of the entire database. Quintus Prolog is a programming language used to interpret the natural language descriptions and then to match them to those descriptions stored in the dictionary. User applications such as the parsing are implemented with minimal difficulty.

3. Natural Language Parsing

A natural language parser to process the description of the image and the sound data was incorporated in the MDBMS in 1989. The natural language understanding capabilities of the parser is limited to captions which are a subset of the natural language. The limited interpretations are controlled by the parser component using the application dependent dictionary as a semantic base. To accomplish the goal of content retrieval of multimedia data, only this limited caption interpretation is required instead of complete understanding of natural language description of the media data.

a. Multimedia Data Described in Natural Language

Retrieval of multimedia data is performed by matching the natural language descriptions with the given query specifications. Based on the Artificial Intelligence (AI) technology of today, if the natural language were unrestricted, then the processing of queries would be difficult to accomplish. The natural language needed to describe multimedia data is very formal compared to daily spoken English. Therefore, instead of natural language descriptions, captions are used to describe multimedia data. Captions are a natural, yet stylized way of writing descriptions within a subset of the natural language. Captions are much easier to interpret and to parse than generic natural language. The important aspect of the natural language is to access entities in a database removing the need to have a complete understanding of all meanings of a word. For a particular multimedia application, the universe of discourse is usually quite constrained. The universe of discourse is the narrow domain of the system's ability to understand everything written in the natural language and the words that are inserted in the dictionary. Nouns are usually concrete objects and, thus, most multimedia databases emphasize still photographs. Other types of fixed time graphics are not usually the objects media databases emphasize, as few verbs can be applied to them and are used to describe the media data. With this in mind, natural language parsing and interpretation becomes a simpler process.

b. Representation Of Media Data Objects

Many researchers in the multimedia field arrange their media data using the Abstract Data Type (ADT). The acceptable Abstract Data Type (ADT) serves a useful purpose in the MDBMS. The media data object models, which in this database refer to sound and image, consist of three parts: the *registration data*, the *raw data*, and the *description data.* A generic example of the media object model's three parts are shown in Figures 2.1. and 2.2. *Registration data* pertains to the display and the interpretation of the

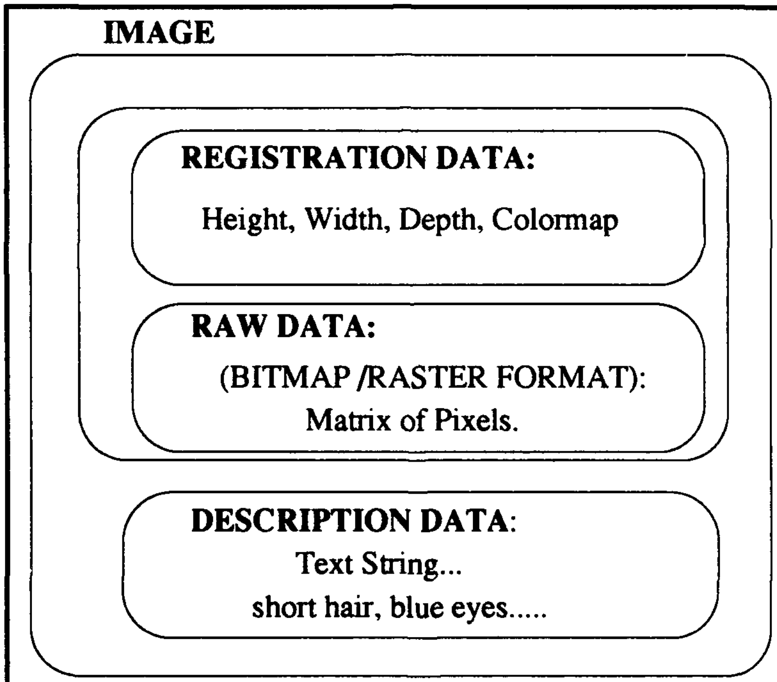

Figure 2.1 Representation of a Value of Type Image

media object. The registration data uniquely depicts which type of media object is stored within the database. Most registration data is of fixed format since the field lengths of information needed to access the media data are known. As listed in the figures, Figures 2.1

and 2.2, the techniques used to encode and to capture the media data give the user a text explanation of the type of media data. The *raw media data* displays the actual media object the system manages. For images, a pixel matrix is one example of a raw data entry. Signals and sounds have bit string representations that are the result of digitizing the original media objects. *Description data* describes the object representation of the raw data in a natural language. This data is used for content search in the computer. A natural language form is entered by the user. After the description is added to the dictionary in the MDBMS, the sentence or phrases are run through a natural language parser written in the PROLOG language. [Ref. 6] The content search is complete and successful when a match is made of the media data description and when the items are entered in the dictionary.

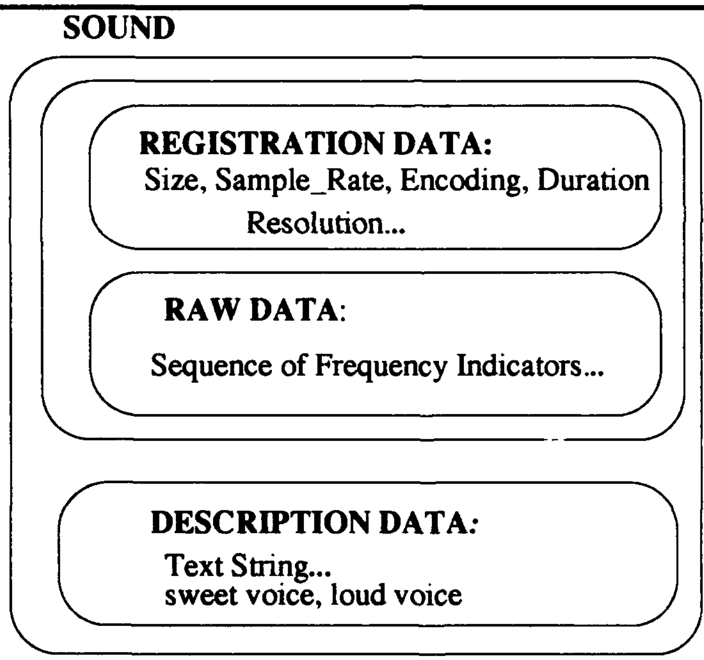

Figure 2.2 Representation of a Value of Type Sound

The relational model is the basis for the design of the prototype of the MDBMS. The relational model is a leading model in the database field in which much research is being done. If a relation has a media type attribute such as sound or image, then a media relation is created for storing the registration and the description data as displayed in Figures 2.1 and 2.2 [Ref. 14]. A separate media relation is created for each attribute with a media data type. A generic relation called OBJECT has the media attributes photo and voice along with the other conventional data type attributes. See Figure 2-3. When an attribute of image type is in the relation, a media relation is created for the image called PHOTO. The PHOTO relation has an integer value called *i_id* that is used as the reference table key number to locate the media relation and link it to the OBJECT relation. Another attribute, *file_id*, keeps track of the path to the raw data where the image object is maintained. Along with the raw data is the description and the registration data. The description data is a natural language description of the image content. The registration data consists of the height, width, and depth of the image object.

OBJECT

O_ID	....	PHOTO	VOICE

PHOTO

I_ID	FILE_ID	DESCRIPTION	HEIGHT	DEPTH	WIDTH

VOICE

S_ID	FILE_ID	DESCRIPTION	SIZE	SAMPLE RATE	DURATON	ENCODE

Figure 2.3 Schema for Modeling Relationships Between Standard Objects and Media Objects

Media sound attributes are handled in a similar fashion as shown in figure 3. The voice attribute of the OBJECT relation has a created media relation called VOICE. The VOICE relation has a s_id that functions like the i_id does for the image. *S_id* is the table key and file_id records the path to the file where the raw data for the sound object resides. The description part holds the actual sound and the natural language description of the sound while the registration data part of the sound holds the variables for recording the sound. The registration data part includes the encoding, duration resolution, size, and sample rate of the sound object.

c. *The Application Dependent Dictionary*

The MDBMS requires auxiliary information from a dictionary. The dictionary is important for parsing and providing every possible natural language word through semantics, parts of speech, grammatical form, and the literal forms needed to represent it. The dictionary is domain dependent since the same word may have different meanings in different application areas. However, conjunctions and qualifying adjectives are consistent in meaning across a wide range of domains. Interpretations from existing natural language systems can be borrowed and included in the dictionary. Some words change significantly between applications. These are generally nouns and some verbs. Because they change in meaning, they must be defined for each application domain. To make matching easier, the properties and the relationships are limited to a small set of primitives. Because we are just looking for the main intent of the English expressions not to capture the media data's full meaning, the following expressions are not defined between relationships asserted by the terms within, inside, part of, containing, including and compromising.

d. *Natural Language Interpretation in the Parser*

The parser translates the text description into a set of predicates called the *meaning list*. By transforming them into a set of predicates, the ambiguity and imprecision of the natural language descriptions are reduced immensely. These predicates state facts about the multimedia data real world entities, like their properties and relationships. First order predicate calculus is the parsing method chosen as the formal representation of the description data. The dictionary is responsible for turning the descriptions into predicates for the parser. The parser's task is to use the dictionary to check the syntactic context to resolve lexical ambiguities and resolve synonyms.

Other features of the parser are the use of supercaptions, a generalization of captions and frames for stereotypical actions. These allow a set of predicates to be derived from terms in the media description. Details of the parser and the predicates are beyond the scope of this thesis but are provided in Dulle [Ref. 6], Keim [Ref. 9], and Rowe [Ref. 14].

The parser is implemented in Quintus Prolog and runs on a SUN SPARC workstation. The parser in its current configuration uses augmented-transition network parsing and interpretation routines. One example of a natural language description and translation into an equivalent set of predicates using the parser is:

Media Description: "She has blue eyes"

Predicates: officer(**x**), component(**x**,**y**), color(**y**).

All the descriptions must be defined in the dictionary so that the correct set of all predicates for the description are recognizable. See Appendix D for a sample of the dictionary. Choosing the right set of predicates is a challenging task similar to knowledge acquisition for expert systems [Ref. 16]. In this thesis, it is enough to state that the dictionary contains all the words the parser can recognize, all the parts of speech associated with any word, and the predicates to use when a word appears in a media description.

4. System Architecture

There are four main areas in the overall MDBMS. These areas are the User Interface, Query Processor, Data Access Subsystem and Intelligent Retrieval Subsystem See Figure 2.4. The *User Interface* provides the user and the database with the interactive information necessary for the query processing to occur.The *Query Processor* accents queries from the user.The Query Processor interconnects the components of the systems by calling the other components when the user queries the system. The *Data Access Subsystem* consists of the Conventional Data Manager and the Media Data Manager. The Data Access Subsystem controls the access to the actual data stored in relational and media DBMS. The *Intelligent Retrieval Subsystem* is comprised of Parser, Generator, Matcher, and Description Manager.

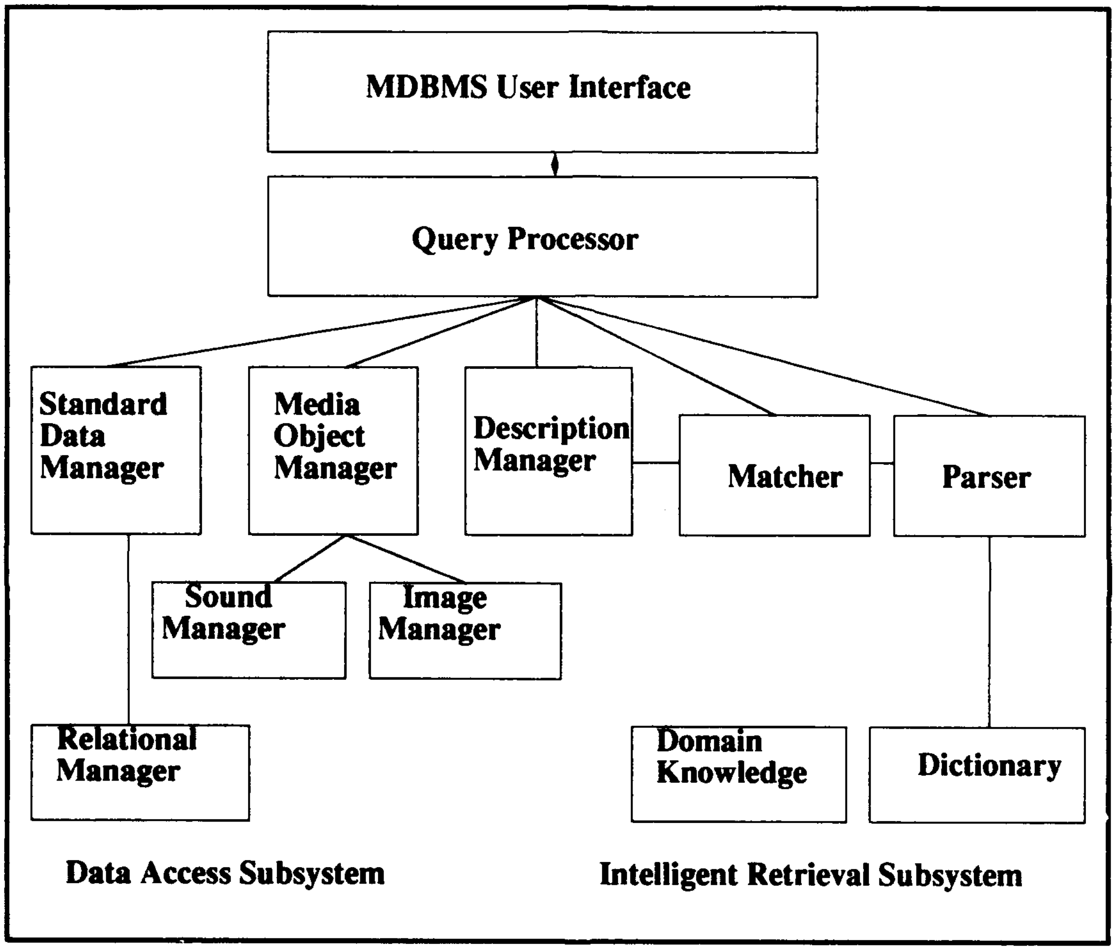

Figure 2.4 The Current MDBMS Architecture

The queries are then executed by the Query Processor which accesses the other components. If a media data value receives a new description, then the query processor calls upon the parser. The parser then must reference the dictionary and create the first-order predicates pertaining to the description. These predicates are then returned to the query processor. The Description Manager receives the predicates which then links the description to its multimedia and conventional data managers. The standard formatted data is managed as in other database management systems by the conventional data manager. Queries related to just the standard data are managed in this region. The *Media Manager* deals with the queries when media attribute types are involved in the user interactions.The Media Manager has two sub-area managers, one for image and one for sound. Lum [Ref. 9] presents the detailed description of the MDBMS architecture. The Media Data Manager handles the processing of both the formatted and unformatted data as media data is not usually in a relation by itself, it usually is part of an relation that includes some formatted data. The Abstract Data Type concept was considered the best method for this model's taskings. Abstract data type is not the same as the usual data types like characters, integer, and boolean since the storage of ADT's are not the same every time the ADT are used. The new media data "values" are handled differently in the processing. The relations with the media data values have an additional relation_id, identifier for the tuple relations.[Ref. 7]

C. IMPLEMENTATION OF THE MDBMS PROTOTYPE

The implementation of the MDBMS prototype became possible with some internal storage structures and the various operation modules. The internal data structures are temporary storage locations while the sub-parts of the user query are attaining the solutions to their sub-query. Along with the internal data structures, a summary for each of the operations (table creation, insert, retrieve and the design ideas for the delete and modification portions) are presented in this section for better understanding of the entire MDBMS.

1. Internal Data Structures Used to Implement the Other Modules

For there to be a need for delete or modification procedure, there has to be a method to have created, inserted, and retrieved the data in the database management system. Internal data structures for temporary storage assisted in placing the necessary data into the proper structures. The data is arranged in tables for storing the catalog information. A detailed description of the create table is provided in Pei [Ref. 14].

The four tables and their functions are outlined below: (1) *Table_List*: The integers are the pointers of the database and the keys for searching, sorting, and maintaining the data located within the other tables by means of the Table_Array. (2) *Table_Array:* Retains the formatted data, possesses the tuple names, and the data if a sound or image attribute is connected with a specific tuples information. (3) *Media_Array:* Tracks the media data located in the database by type, be it sound or image. (4) *Att_Array:* Contains the attribute names and data types for both formatted and media data for every attribute. It is an interface of the media (image and sound) data.

A fifth array, called a *Value_Array*, exists in this database arrangement. This array provides the valid value types of the attribute entries as they are entered in the attribute array. The values are the traditional char, integer and real plus the special value of media data. These items become very important when attempting to relocate a specific file or piece of data for modification. [Ref. 10, Ref. 15]

2. Table Creation

Table creation is one of the major operations in MDBMS. The user defines the template for the data storage structures based on the arrays listed in Section C above. The acceptable table name size, values, and categories of attributes necessary for table creation are given to the user in order to create the data table relations. The user must place the relation name, attribute name, and the data type of each attribute in the system. The information is captured and stored in the MDBMS system tables. An active media list is generated and maintained to keep track of the media ADT s created and in current use. [Ref. 14]

3. Data Insertion

After the data storage structures are defined, the information the user wants to store is then inserted into the system. This operation is the first operation where data values are placed into the MDBMS prototype. The data insertion procedures are explained in detail in [Ref. 14]. The five types of data supported by the prototype are character, string, integer, float, image, and sound. Three of these are formatted and two are media data types in MDBMS. Five value arrays are designed for data insertion. The value arrays are discussed in Chapter V. These arrays function as temporary tables. Tracking the pointers in the value arrays is very important to the modification portion of the database. Replacing the data stored in the correct relation requires the knowledge of the pointers' locations.

Data is added to the database by creating relations and placing the data in the relational structure identifier tables the user created in the table creation procedure. Each attribute is checked to ensure that a media type is not encountered. If a media type is encountered, special processing must occur.

a. Formatted Data Insertion

Placing conventional formatted data in a relational structure is simple. The relations structure used requires the attributes and the attribute data types to be clearly defined. The relation name, attribute names (in sequence), and the attribute data type must be integer, real or character. Each attribute data type must be followed by its length for exact definition. When standard formatted data types alone are used, which means media data is not involved, the standard data and the create table command are passed directly to INGRES. This is accomplished by means of the create table command in SQL. Because the media data type cannot be handled by the INGRES system directly, media data inclusions follow another procedure for getting added to the database.[Ref. 16, Ref. 18]

b. Media Data Insertion

When media data type attributes are encountered, the applicable media type, audio or image, is entered in the Att_Array Table under the appropriate data type value image or sound. A request to INGRES to create a media relation specifically for this attribute is processed. The structure of the media relation differs depending on whether it is image or sound. *I_Id* is the system assigned internal identification that is used for each value to be entered in the media image table. For the sound data type, the *s_id* is the system assigned internal identifier that is used as the value to be entered in the attribute *voice* of the user relation OFFICER. Both are integer indexes in their respective tuple relations.

For example, photo, an attribute of the user specified in the relation OFFICER is arranged in the database in the following files. *I_image_f* is the file where information on the image and the exact path where this file exists. *I_image_d* is where the natural language description of the content of the image. The parameters of the height, width, and depth for the image file are necessary to reproduce the recorded image data. For the sound data type, the s_id is the system assigned internal identifier that is used as the value to be entered in the attribute "voice" for the user relation OFFICER.

Frequency, sample, resolution, and encoding are the parameters that are necessary to reproduce the recorded sound data. The way that the audio and image data are added to the database requires the integration of the IBM PC and SUN systems. The last attributes for any new relation are the media types image and sound. This provides the indexing and pointing necessary to attain access to the media storage of the database. Another example: if a user desires to add the additional data of an image to his OFFICER relation, the user must first digitize the image in the correct format on an IBM PC for the system to display back on the SUN system at a later date. At the present, a video camera recorder is used to record the photo that is then converted to GIF, Graph Interchange Format, on an IBM PC file. From the IBM PC, this digitized image is then transferred to a SUN workstation where the GIF file is converted into the SUN raster format [Ref. 15]. The image the user wanted in the OFFICER file is now available for use modification and/or display by using the SUN workstation. A more in depth discussion is presented in [Ref. 15].

For audio, the data insertion is entered in the same manner. The difference is that the sound file is stored in the PC instead of the SUN workstation [Ref. 1]. No duplication of the sound file is necessary. In the MDBMS, a text file is generated to keep the information of the sound by giving it a file name, taking the registration data of that sound object to include parameters of frequency, resolution and the duration. This file assists in locating the actual sound maintained in the IBM PC. The majority of the parameter information is decided before the actual recording of the sound file datum and is stored in a way that simplifies the processing of sound in the MDBMS. The *active_media_list* discussed in the create table module is used the processing of the media data insertion as well. For accessing data for the modification procedure, understanding the proper manipulation of the active_media_list counter is necessary.

4. Retrieval of Data

Retrieval is the most important operation in the MDBMS. The deletion and modification operations depend on the proper retrieval of data from the database prior to commencing the removal or the update of the queried data. Simple retrieval was discussed in [Ref. 15]. More complex queries that support nested conditions and multiple selections are presented in [Ref. 2]. For this thesis only, basic retrieval queries are used for demonstrating the deletion and modification operations the on MDBMS prototype.

a. Retrieving Formatted Data

Retrieving formatted or standard data in the MDBMS is precisely the same as in any other database management system. When media data is not involved in a query, the user request is passed directly to INGRES. INGRES handles the request by retrieving the information from a temporary table for the user to review and then returns the data by erasing the temporary table.

b. Retrieving Unformatted Data

If the retrieved data contains a media attribute, the query is broken down into multiple sub-queries to a level that the INGRES embedded SQL can decipher. It is divided into two parts: one part involving standard data and the other part involving media data. Each sub-query part is processed individually by either using the natural language parser or by passing to INGRES directly. Both result with pointers to the qualified data. The pointers are combined for accessing the corresponding standard and media objects.These results that have been reconstituted provide the user with the final result for the given user conditions. This is a difficult maneuver if a media image is actively part of the user request. By outlining sub-queries so INGRES can handle them, the user request gets processed more efficiently [Ref. 15].

5. Modification and Deletion Requirements in System Design

Because of the hardware configuration and the software used in the NPS MDBMS prototype, special structures in both software coding technique and the storage data structures are necessary. There are three areas to consider when describing the update and modification portions of the database management. They are modifying straight standard data, modifying media data with sound, and modifying media data with image.

Queries to standard data are passed directly to INGRES to process. No special requirements come into consideration until media (unformatted) items are involved in a query [Ref. 13, Ref. 15]. To process the multimedia data, some items must be repeated in the catalog tables. This, unfortunately, can cause some inefficiency and inadvertent errors.

For updating the sound, a rerecording or rewriting of the description is permitted. Each time the audio portion is pulled from the tables for a query, all the recorded sound file segment attributes are revealed. These must be checked and can be changed. The description and text that accompanies the actual audio in the MDBMS can be altered without listening to the sound data. Rerecording the sound or changing the speed, frequency, or other key attribute values does, in itself, change the present sound sample.

The media data of type image requires several preliminary steps before combining the sub-query answers and reconstituting the final response to the users initial request. The temporary tables and their use were touched upon earlier in this chapter in the create table, insert, and retrieve sections. The concept of temporarily holding the user supplied information until verification and then passing it to INGRES for the proper storage are executed in the same manner for the deletion and modification processes. The completion of the modification and update procedures provide the MDBMS users with a total database management system. This MDBMS then allows the users the capability to correct data errors already inserted in the database and remove superceded data information stored previously.

III. MODULARIZATION

The modularization of the MDBMS prototype was performed by a team effort. As a result of the joint work, the written discussion of the modularization is also joint. Hence, this chapter also appears in total content in [Ref. 2].

A. DEFINITION OF MODULARIZATION

We used two software principles to assist in the development of the MDBMS prototype: the concepts of modularization and version management. Both will be discussed in this chapter. Modularization is a way to structure programs. It is called modularization because the process is to break the program down into smaller parts called modules. A module usually consists of a collection of related programming-language entities such as procedures and data types. Modules are designed small enough to be understandable by themselves thereby enhancing the clarity of the program. This is important to software engineers and users. The different modules of a program are in relationed to each other resulting in a module hierarchy. Modules on a higher level of the hierarchy use functions or procedures of lower level modules. Large projects with numerous people working on them prefer this method of programming since modularization lends itself to allotting each individual their piece of the program in which to be creative without interfering with others.

For the version management, we used the Source Code Control System (SCCS) provided by the UNIX operating system. A version manager is very beneficial in a continual software project that will have people working on the programming long after the original programmers have departed. The initiation of both modularization and version management the increased efficiency in developing the MDBMS prototype.

B. REASONS TO MODULARIZE

There are principle reasons for modularizing a program. It makes the software development process easier because modules are more understandable. A program which does not consist of substructures is hard to understand. Dividing a program into modules makes it easier to follow. The modularized programs are much easier to write as work can be divided among a team of software engineers which reduces the time to complete the final product. Modularizing also reduces testing time, as an established hierarchy aids in developing a testing pattern. Testing of modules is easier than testing a program code without any structure. Modules can be altered more quickly than an entire program if changes and improvements are added. When the need arises to change the program code, changing one or a few modules will be enough to get the desired result. When a new system is to be built, it is possible to reuse modules of a previous system if the same or similar functionality is needed in another program. In the following section, the advantages of modularization is discussed in detail.

1. Comprehensibility

A program without delineated structures and substructures is difficult to follow. Over time, it has been proven that modular programs possessing high cohesion and low coupling results in easier-to-understand systems. A module is highly cohesive if the functions and objects within the module are closely related to one another. Low coupling refers to the manner that each module interacts with each other. The smaller or narrower the interface is, the better the system. With the interface minimized, changes to one module will not effect the entire system less those interfaces between that module are too great. A module hierarchy allows the user to follow the program and see how the modules interrelate and to understand how each module effects the other modules.

2. Division Of Work

To complete a large task in a reasonable time, it must be divided among the people participating in the project. This can be done using modules as basic units. When more than one person is working on a program, dividing the work into modules increases the efficiency in the production of the program. Each module given to a person of the project should be small enough to be implemented within a relatively short period of time. Programming team members can work independently on their modules. Each small part of the program should run independently. This arrangement also makes the team work run more smoothly. A team member is able to compile his work in shorter time periods. Working modules appear more quickly, with completion of debugging modules in smaller time increments. If the implementation of a module takes too long, then it is necessary to break it into smaller pieces.

3. Easier To Write

Modules are smaller pieces of code to work with which means they are created more quickly and implemented easily. The smaller parts are then built upon as a larger section of related code. This modularized approach lends itself to work that can be divided. By writing programs in parts, the testing takes less time and the program becomes much more adaptable to change and discovery of errors.

4. Easier To Change

Change is a fundamental characteristic of software systems. Improvements in software and hardware occur frequently and quickly. Users may ask for new features or changes to old ones; the system may move to new hardware or a new operating system; bugs may be discovered during testing; and/or performance measurements may show bottlenecks. Costs of these changes can be enormous if entire programs must be rewritten to meet the new equipment or software requirements. A modular format lends itself to change with minimal effort. If better ideas for program functioning are incorporated in the program, the modular structure allows interchanges of the units with minimal restructuring. Changing a module instead of an entire system is less costly in may ways. The building block formation can use general interfaces in order to achieve the connectivity necessary to have the program run smoothly. Overall, to modularize is to achieve better quality programs more efficiently and in less time.

5. Reusability

Another advantage of modularization is that some of the modules can be reused with little or no restructuring of the base module. When you build a new system, you may need the same functionality that you used in a previous system. An example is a module providing advanced string or window operations. This module can be used without changes for many applications. Instead of rewriting these modules, it saves much effort if you can reuse modules of a previous system.

6. Easier To Test

Having smaller parts to work with facilitates debugging and testing of the capabilities of the modules. In a large system, each module should be tested individually and large collections of modules should be slowly built up to test the whole system. Since modules are much smaller units than the whole program, errors can be found faster. Debugging is the process of locating errors, defects, or problems that effect the outcome of the module. Debugging often involves a lot of detective work. Testing is working the system module by module for accurate and predictable answers. Testing has much broader scope and usually assumes you have finished most of the debugging. Testing may uncover problems which may lead to more debugging as the programmer may discover other flaws in the software.

In addition to these advantages, a modular system provides other improvements to software programming projects. Modularization can be used to achieve two basic principles in systems development: information hiding and abstraction (or encapsulation). Information hiding is the software designers decision to "hide" data structures and operations that the user does not need to know. Some reasons for information hiding are to prevent data structures details from being revealed to the user, keep algorithms that perform operations from being shown, or protect the details of an interface to an operating system. Information hiding and abstraction focus on different aspects. Information hiding focuses on what to hide; abstraction focuses on what to reveal. Information hiding tries to protect you from change you ask what design decisions might change, and arrange to hide them so you cannot depend on them. The sorts of decisions one might hide include the algorithm for carrying out some operation or the representation of some data structures. Other examples are the details of an interface to an operating system or to special purpose hardware, or the policy for allocating some resource ordering certain operations. Every abstract data type is an information hiding module; however, a module may not be an abstract data type. A module can also be a set of unrelated functions. An abstract data type module provides a collection of procedures for manipulating the encapsulated data structure. For example, a module might hide the representation of a stack. It would provide operations for pushing elements onto a stack, popping elements off the stack, reading the top element of the stack, and initializing the stack. These four functions are called the interface procedures of the module.

C. HOW TO MODULARIZE

There are steps to follow in order to modularize a program or system. First, the major sections of the design decisions and the sub categories located within those sections must be identified. The major groups become the top levels in the generic module hierarchy. A *decomposition record* is maintained that keeps track of the rearranging and regrouping of the system into modules. This decomposition record is important. When the design decisions call for information hiding, the decomposition record keeps track of all the hidden information. Then the project secrets are accessible to software engineers. As mentioned earlier, the size of a module should be manageable by one person. If the module is too large, then it can be further divided into smaller parts. However, one person may have more than one module on which to work.

The program structure is reflected by a module hierarchy. The module-hierarchy shows the dependencies between the modules. The boxes represent the modules. A top-down design displays relationships of the modules to one another by the line links or connections. Figure 3.1 shows the dependencies of the modules. Module 1 calls three modules on the next level of the hierarchy, Module 2, Module 3, and Module 4. They in turn call on modules they are connected to on the next level down.

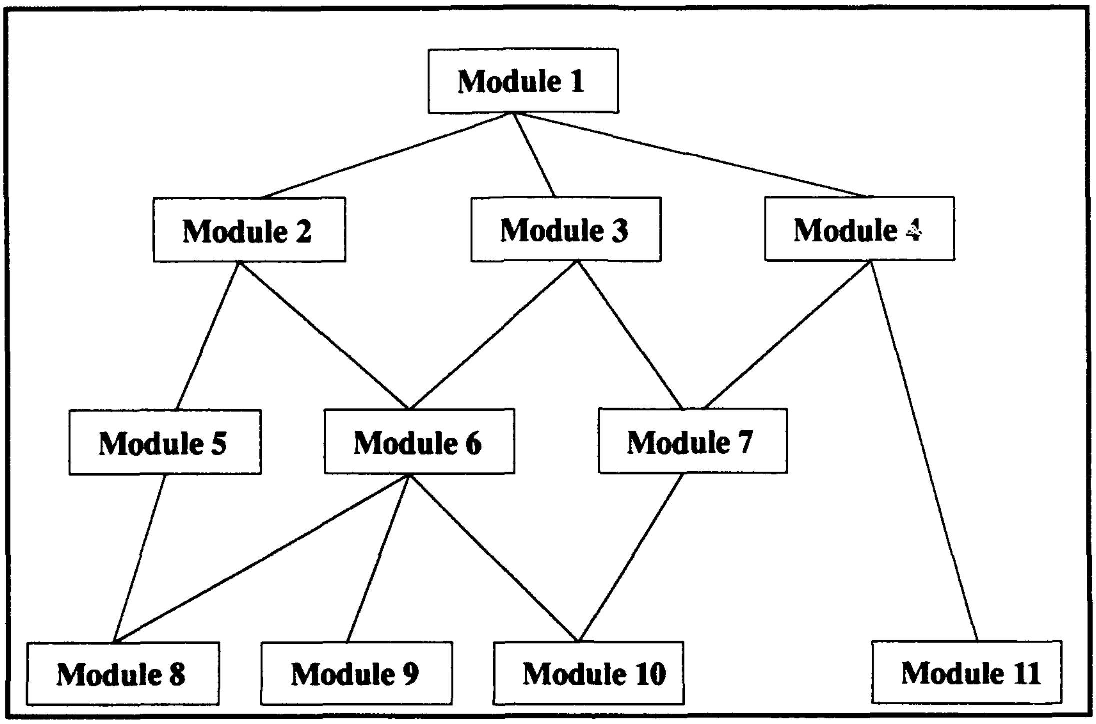

Figure3.1 Generic Module-Hierarchy

Another document, entitled the module *dependency document*, defines the module dependencies and a module hierarchy for a given system (see Figure 3.1). The system module dependencies are shown by the line connections. The *program document*, after the division process is complete, resembles separate modules. Now the dependency relationship with other modules is identified. These are recorded as import and export interfaces listed in the documentation of each module (see Figure 3.2). An *import interface* is the listing of all functions called from other modules that are necessary to perform the modules given task. The *export interface* consists of the functions provided by the module in question. Figure 3.2 is an example of how each module in the hierarchy is structured.

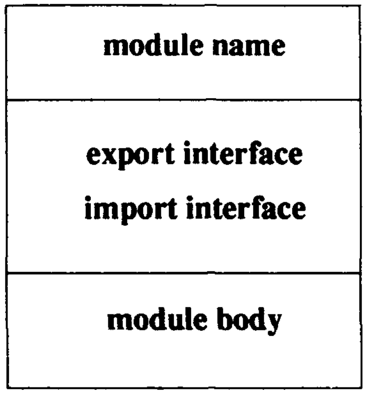

Figure 3.2 Generic Module Structure and Format for a Module

```
export interface:= export <function 1>, <function 2>,..., <function n>;

import interface:= import <function 1>, <function 2>,..., <function i>;

                   from module <module name1>;

                    import <function 1>, <function 2>,..., <function j>;

                   from module <module name 2>;

                   import <function 1>, <function 2>,..., <function k>;

                   from module <module name m>;

module body:=implementation of the exported functions by using the

                   imported functions

                   (including not exported and hidden data structures)

                   functions
```

Figure 3.3 Example of Import-Export Preamble

D. MODULARIZATION OF C PROGRAMS

The MDBMS Prototype systems are implemented in the C programming language to include the MDBMS prototype. Unique advantages and disadvantages exist with any programming language and some of both will be discussed in this section. In C, there does not exist viable support for the principle of modularization.

1. No Support For Modularization in C

The C-programming language does not support modularization. To write modular programs in C, the code has to be subdivided into separate parts which are stored in separate files. Because everything is listed in separate files, a programmer may perform separate compilations or can use the include mechanism. Using separate files does not reflect a modular structure as obvious recognition of where functions are imported from or exported to is not blantantly displayed.

Other languages such as Modula 2, ADA or object-oriented languages like Smalltalk or C++ do support the modular format better. However, these languages are not available for use for the completion of this MDBMS prototype. Transferring an existing program from one language to another is not a viable option as it creates great amounts of overhead. Therefore, in order to achieve the advantages of modularization, a concept for modularization in C was developed.

2. A Concept For Modularization In C

Organizing the program by dividing the data structures and functions in separate files corresponding to their purpose gives some of the benefits as a program constructed in a language that supports a modularized format. As mentioned before, C files can be compiled independently and larger C programs can be divided into more manageable and smaller sections to emulate the benefits of modularization. Similar functions, previously separated, can often be condensed into more general, parameterized functions thereby gaining clarity, uniformity, and reusability. Since C files can be compiled independently, it is convenient to partition large C programs into smaller and more manageable parts to achieve some advantages of the modularization concept. Independent compilation allows files containing C program components to be checked separately for syntactic and semantic errors. Moreover, when a program is modified, it is only necessary to recompile the effected components. The Unix utility *make* is used to automate this process. This special option make, is a recursive call to update files according to their dependencies. Make can review all the separate files after one or more have been altered and define in which order the effected files must recompile. the make utility is explained in detail in Appendices C and D.

Another mechanism used to achieve some kind of C modularization is the *include* mechanism. Arbitrary files can be textually included in a C program by means of the include instruction. The capability to include files textually in a program allows common constant, data, type and function declarations and definitions to be kept in separate files. These common declarations and definitions can then be used in all parts of the program. Keeping common declarations and definitions in separate files and then including them in C programs is a popular style used for writing C programs. Examples are constant data, type, and function declarations or definitions, and the standard input/output declaration file stdio.h and math algorithms. As mentioned before, although the C language does not support modularization, we can achieve the following advantages can be achieved by dividing the existing program code into separate parts:

- Program modules can be developed independently.
- Changes to the program can be done by only changing single modules.
- Clarity of design and structure.
- Program code is easier to understand.
- Maintainability.
- Reusability of modules.
- Uniformity.

a. Similarities Between C and Modularization

Separate files of a program can be developed independently as can modules. Having the program code in separate files, the program is easier to understand and provides a better clarity of design than a program without any structure. The import/export information contained in the files gives the impression of a modular hierarchy. As improvements or changes occur in the program, single files or modules are all that are effected. A revision of the entire program is unnecessary. Separate files are maintainable and reusable just like modules. The uniformity of the files with the import/export interfaces defined is another positive similarity to the modular format. Through the division of an existing program code into separate files, C cannot fully support the modular format but does possess some important characteristics of modularization.

b. Making Dependencies Between Modules Visible

To make the dependencies between the different parts of a modularized program visible the export and the import interfaces have been added to the separated files. In C, they should be put in the documentation header of each file. If all modules are compiled separately, it is only necessary to recompile the module someone is working on and link it with all imported modules to get an executable program. A list of all necessary files used for the system should be available, making the module hierarchy visible. For each file, the purpose and its place in the module hierarchy. Figure 3.4 shows the style of documentation we used for the MDBMS. The export interface of the module shows the functions that are called by other modules, which modules they are exported by and which module this function is imported. The insert_tuple() function is exported to the Delete.c module.

The import interface displays the functions that are brought in from other modules.C files can be used to partly implement data abstraction and information hiding. An abstract data object, as defined in the previous section, is an object that can be manipulated using only the operations supplied by the definer of the object. The user cannot directly manipulate the underlying implementation of an abstract data object. Details of how an abstract data object is implemented are hidden from the user. Hiding the details prevents the user from: making programs dependent on the representation. The representation of an aostract data type can be changed without effecting the rest of the program. For example, the abstract data type set may be initially implemented as an array, but this representation may be changed to an ordered list later on for storage efficiency. Integrity of abstract data type objects is preserved by forcing the user to manipulate these objects using only the operations provided by the designer of the abstract data type. Examples of abstract data types are stacks, queues, sets, databases and binary trees. However, a C file is not a true data abstraction facility, because it only partially supports data abstraction. If you link an independently compiled C module to some other parts, you can not prevent the user from accessing all functions and even the internal data structures of other modules.

```
*********************** InsertModule.c ***************************

    Title : InsertModule.c
    Author : Su Cheng Pei
    Date : November 15, 1990
    History :
    Description : This module implements the insertion process in the Multimedia
                  Database System.
    ***************************************************************
    Export Interface :
    print_all_table() :Prints out the table catalog information on the screen.
    insert_tuple() :Inserts a tuple of a particular relation.
    display_tuple() :Displays the tuple before insertion.
    check_media_description():Checks the media description by connecting
                  to the parser.
    ql_insert_tuple() :Translates SQL statement to insert a standard tuple.

    ***************************************************************
    Import Interface :
    get_sound_value() :Gets a sound value of a media attribute from the user input.
    yes_no_answer() : Gets yes or no answer from the user.
                   from UserInterface.c
    check_table_name():Checks if the table name is duplicate.
    get_media_name() : Gets media table name by appending table_key at the end of
                  att_name.
                  from CreateModule.c
```

Figure 3.4 Export/Import Interface of a Module

E. MODULARIZATION OF MDBMS

The MDBMS prototype is implemented in the C programming language. The majority of the original program was in a single file and was very difficult to follow. Three students were working with the prototype. Considering the size of the program and that multiple students were working on three different parts of the project, namely complex query processing, graphical user interface, modify and delete, it was decided to modularize the MDBMS prototype. Before beginning work on the modification and deletion sections of the MDBMS, the program was first broken into large chunks of code in order to understand how the program worked. When modularization of the MDBMS prototype was begun, the module hierarchy was as shown in Figure 3.5 The existing program code into separate files corresponding to their purpose. Then each part was divided again until the final module hierarchy was obtained.

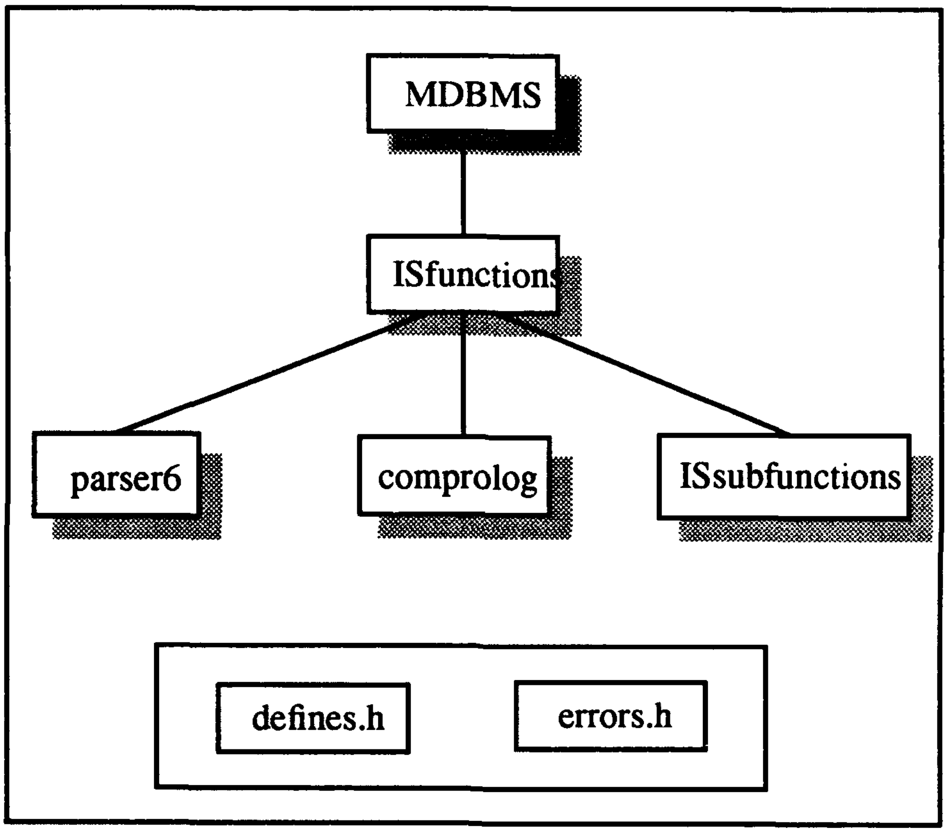

Figure 3.5 Existing Module-hierarchy

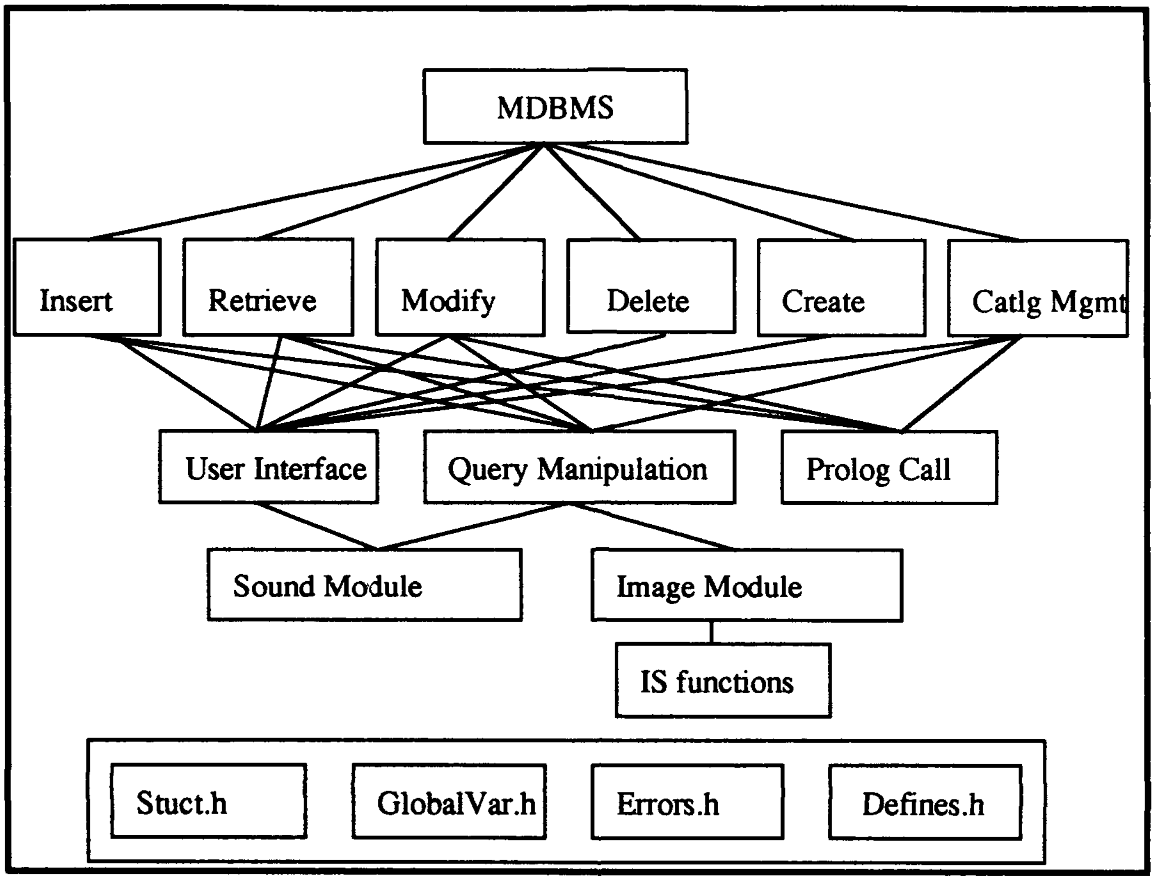

Figure 3.6 Current MDBMS Hierarchy

The resulting modular format is shown in Figure 3.6. The main program calls the functions and routines imported from the modules to make the system operate. Modularization of the MDBMS was not completed as we could almost spend the entire thesis on modularization. Instead, we incorporated large basic modules containing related functions and procedures. The MDBMS module hierarchy now consists of 6 levels. In Figure 3.6, each box represents a module. For instance, the module 'MDBMS', which is the main program calls the modules 'Catalog Management','Create', 'Insert', 'Modify', 'Delete', and 'Retrieve'.In the diagram, the straight lines show the dependencies between modules. Although the module 'MDBMS', which is at the first level of the hierarchy, calls some of the modules on the third level of the hierarchy, the dependency is not shown on the diagram.

There is, however, still much to do on modularization. Considering the time needed to complete our parts the complex query processing and modification working on modularization was stopped at this point. To make the dependencies between the different parts of the MDBMS program visible, import and export interfaces were added in the documentation header of each module (Figure 3.5). Functions that can be used by other modules are placed in the export interface sections of each module. The export interface therefore summarizes the functionality provided by a module. For example, the function print_all_table() taking place in the export interface of 'Insert' Module can be called by the 'Retrieve' Module.[Ref. 7]

In the import interface of each module, functions that are called from other modules are mentioned. For instance, the function get_sound_value() taking place in the import interface of 'Insert' Module, is called from the 'User Interface' Module. In addition to our work on modularization, we also used some helpful tools provided by the UNIX system were used, namely *sccs, lint, make*, and *dbx*.

F. TOOLS USED WITH MDBMS

Some UNIX tools were used to improve the development of the MDBMS prototype. The three presented are lint, dbx and the SCCS version manager. The fourth, make, is discussed in Appendices C and D of this thesis.

1. Lint - A Program Checker

Lint is one of the UNIX operating system utilities. The lint command is incorporated in the make file of the MDBMS program to automatically assist in the debugging and discovery of the prescribed errors. Lint attempts to detect features of C program files that are likely to be bugs, non-portable or, wasteful. Lint performs stricter type checking than the C compiler. It runs a preprocessor as its first phase which allows certain questionable code to be altered or skipped by lint. Its second pass checks the files for compatibility and consistency. Some of the main items lint manages to find are unused arguments and variables, inconsistent function calls, and ignored function returns. During the compilation of C programs, lint helps to find unused arguments, unused variables, variables which are set but not used, inconsistently used function calls, and always ignored function returns. The lint command was put in the Makefile so that during compilation, lint is invoked automatically by the Makefile. The make utility, which is a command generator, is explained detail about 'make' and its use in Appendices C and D.

2. DBX -A Debugger

DBX is a utility for source-level debugging of execution of programs. Run time errors during the execution of a program can be found using dbx. To run dbx, type "dbx <executable file name>". To run the program in dbx, type "run". dbx. Dbx is a source-level debugger. One of the options of dbx is the trace command.To trace in a function, type "trace in <function name>". This allows a user to run the program and trace in a function that may not work properly. This tool was used to find some run time errors during execution of a program.

3. SCCS -A Tool for Version Management

Another item that is very important to team programming is the use of a version manager. We chose the Source Code Control System (SCCS) provided in the Unix system which possesses various utility programs to allow more than one version of a file. Programmers have access to the latest version or return to any previous version of a module. All changes are stored as a new version number and can be accessed at any time a question may arise about the changes in a program module or procedure. Using the SCCS worked well for this project, as each of the team members had a stabilized version of all modules except the one(s) that the individual was modifying. If you want to try another way to implement a module, you can work on a version of that module while you still keep the older versions. You can always go back and work on any older version you want. As I worked on the modification portion of the MDBMS, I maintained a copy of a retrieve module that performs basic retrieval operations. Other team members were able to continue to improve upon the existing retrieval module without interfering. I could also test another way to code the modify operation and work on it but still have the older versions.

SCCS is ideal for team projects that require access to overlapping code and historical changes. The cataloging is very accurate, easy to use, and quite space efficient for database storage as only the changes are saved with each version that is created and stored. To use the SCCS, a directory must first be created and given the name SCCS. Files must already exist that you want to start version managing. Some basic commands for use of the SCCS are create, delget, edit, prs, info. Examples of how to use them are shown below. To create the initial version of a file, one must type "sccs create <filename>" at the prompt. You should not be in the SCCS directory at this time. If you want to work on this version and still maintain a pure copy, then the command "sccs delget <filename>" is what should be used. What you have done is created a version number for the initial version and set up another copy of that version for editing. It is not available for editing until the command "sccs edit -r<version number> <filename>" is typed. The version number is shown when a delget is started. The -r option is used in case the desired version is not the latest version. The "r" stands for revision. An example of editing a previous version is if I want to edit the InsertModule.c version 1.6.3.4. I would type "sccs edit -r1.6.3.3 InsertModule.c". To see how many versions of a particular module exist,.the command "sccs prs <filename>" is the way to access that information. The command "sccs info" shows exactly which versions of all files are currently being edited.

IV. DELETION IMPLEMENTATION

My ideas for solving the deletion problem began with designing and/or enhancing functions and procedures that are compatible with the currently working modules and procedures in the MDBMS system. The MDBMS prototype architecture outlined in Chapter II and modularization discussed in Chapter III were the first steps towards building a complete system. Three students commenced work to complete the basic operations of the MDBMS operations not currently implemented and to enhance the quality of some prototype operations already running. We continued to modularize, incorporate complex query handling, modify the user interface, and develop the deletion and modification operations [Ref. 2, Ref. 15].

My design makes use of existing data structures and implementations for some of the underlying retrieval operations. Additional data structures to capture the necessary user information are found in the modification operations for the formatted and media data. By working with the existing structures, integration of my procedures into the MDBMS was less cumbersome. An overview of the design, discussion of the implementation, and examples of the deletion operation are in the following sections.

A. INTRODUCTION

Design issues for the modification of information stored in the MDBMS were first discussed in 1988 when INGRES was the DBMS chosen to build the MDBMS. The catalog management system is very important for all these major operations in the MDBMS prototype. The restrictions with the INGRES version to implement the MDBMS prototype have been mentioned in Chapter II. Newer versions of INGRES have eliminated some of the restrictions; however, all the prototype code would have to be rewritten in order to work with a newer version of INGRES. The SUN corporation improved its SUN workstation. Current SUN models directly support sound functions. The prototype source code would need restructuring and substantial recoding if the new hardware and software were purchased. Instead of making these investments, the original configuration of the prototype with an IBM PC as a backend server for sound data management in the MDBMS, would continue. The IBM PC is connected to the SUN workstation via the local area network, ETHERNET [Ref. 1]. The constraints of the hardware structure and the older versions of the software due affect the design and implementation of the deletion procedures. However, since this is only a prototype system and not one for the commercial market, my work was to complete the deletion and modification portions of the current system structure with its current design. The principles involved with these operations can be demonstrated on this prototype without the costly upgrades.

1. Why Deletion Operations are Important

Deletion of outdated or no longer needed information is one of the basic processes expected in every database management system. In order to perform the deletion operation, the user must already have created a table (i.e., defining the relations), inserted some tuples of data and stored that data within the database. The user can now remove any extraneous data by means of a deletion operation. A retrieval of the tables or files where the information is stored is first performed. This is executed based on the user's requirements and the relational conditions placed on the database retrieval system. After the conditions are provided, the database searches for the given conditions. Once the data are retrieved, removal occurs. An in depth, step-by-step example of removing media and formatted data in the MDBMS is found in Section C of this chapter. Formatted data removal is the easiest method of deletion to perform. A method to remove media data in the multimedia systems is more complex to remove for a database because of the storage requirements. For deletion, use of the integer table key and the integer value of the media type provide the user with the ability to access the proper media table during the execution of the procedure.

2. Schema of Database Relations

The MDBMS database relational schemes are examples of a practical way to store information about a U.S. Navy fleet. Records maintained on the ships, weapons, officers, missions of the ships, and ship homebases are common items that must be accounted. The information stored in the database on the objects listed above are shown in relational form in the Figures 4.1 through 4.3. For a ship, the name, number, type, year it was built, displacement, current mission, captain and executive officer are important pieces of information to access. For a record on weapons, the name, type, range, power, and description are key items to store on weapons. Officer files are another set of files that should be stored in a fleet database. The officer name, rank, identification number, salary, reporting year date, along with a current photo and recording of his voice are items the fleet should have available for review. Base ship home ports listed by name, location and size are other key pieces of information. Mission assignments with the directions, goal and task outlined provides the fleet an accounting of the total area of involvement for members of the fleet at any given time. See Figure 4.1.

The primary keys are user defined and indicated by the underlines. The media data types are the abstract data types defined in the MDBMS prototype. Each media attribute has an integer associated with the location of the separately stored media relation. As described in chapter II, media objects are created and stored as the media data types image and voice. These media relations are not visible to the user. Ship is the first relational table created in the MDBMS. An example of how the media storage table is created for the SHIP relation is as follows the table key for ship is "1" as it is the first table in the database. The media relation table for the ship photos is called PHOTO1 to correspond with the SHIP table key number "one". In Figure 4.2, the media image attributes added for each SHIP tuple relation are placed in the created media table PHOTO1. For each of the ship tuples that contains an image attribute, it is stored in the media image table PHOTO1. The image identifier (i_id) which is an integer acts as index. The i_id for each media relation will indicate where the particular SHIP image is stored. The WEAPON table is the third table in the database; hence, PICTURE3 media relation table is created for the weapon relation pictures. Another example is the fourth table in the database is OFFICER. The two media relation tables created for the OFFICER.photo and OFFICER.voice are media tables

PHOTO4 and VOICE4 respectively as shown in Figure 4.2 and Figure 4.3. Figure 4.3 is used to explain the design and implementation of the deletion process.

B. DESIGN AND IMPLEMENTATION OF DELETE

The key to the deletion and modification procedures of the MDBMS is to understand how each of the different tables are located and manipulated. Gaining access to the relation conditions is simpler in theory than in practicality with the MDBMS configuration. Data,

SHIP

s_name	s_no	type	yr_built	disp	m_id	b_id	capt_id	exo_id	photo

SHIP_WEAPON

s_no	w_name

WEAPON

w_name	type	fire_range	power	picture

OFFICER

o_id	o_name	rank	salary	rep_yr	photo	voice

MISSION

m_id	m_name	direction	goal	task

NAVY_BASE

b_id	b_name	location	size

Figure4.1 Navy Ship Database Relational Schemes

once stored in the system must be retrieved, stored in a temporary location, manipulated, and then the final, newly acquired resultant data returned to the original table in the database system. Some code revisions to get the deletion process operating are given in section C.

PHOTO1

i_id	f_id	descrp	height	width	depth

PICTURE3

i_id	f_id	descrp	height	width	depth

PHOTO4

i_id	f_id	descrp	height	width	depth

VOICE4

s_id	f_id	descrp	size	samp_rate	encoding	duration	resolution

Figure 4.2 Media Relation Schemes for Navy Ship Media Attributes

Some of the procedures require user supplied conditional information. This user information sometimes must be restructured so that INGRES can perform operations in formats that INGRES can manipulate. The work began with methods to delete formatted data. This is the easiest type of information to delete, as it is in conventional coding format. Because formatted data is handled in a data type directly accepted by INGRES, formatted data can be retrieved, deleted, and the table reconstituted for storage in the database with minimal effort.

With the aid of Figure 4.3, we will talk through the insertion, retrieval and deletion of data for the table relation OFFICER. The table contains the attributes o_id (integer), o_name (character 20), rank (integer), salary (float), rep_yr (integer), photo (image), and voice (sound), created during the initial execution of the create table operation [Ref. 14]. INGRES separates the creation into three parts to set up the relation OFFICER, one for the formatted Officer relation, one for the image media relation and one for the sound media relation. These are shown in Figure 4.3. The suffixes on the media tables have been explained earlier. The input data to the attribute photo and voice in the OFFICER relation are integer type and are used as the indexes to the i_id of the PHOTO4 relation and the s_id of the VOICE4 relation respectively. Data for Jeff Kulp is added and the values stored temporarily in the data value arrays prior to INGRES insertion. He is the first entry in the relation OFFICER. When the user prepares to insert a media data attribute value, the image or sound filename must be input at this time. In the case of image, "n/virgo/work/mdbms/gif/jeff.ras' is typed into the system. The system inserts that filename into the field f_id of the relation PHOTO4 which represents the media attribute photo. All the data pertaining to the file, height, width, depth, are extracted from the header file and inserted into the relation PHOTO4. The i_id index is assigned (which is 1 in this example) and then the i_id index number is placed in the relation OFFICER photo attribute column. The system inquires as to the description of the photo which is "big head". This is inserted into the "description" attribute column of the relation PHOTO4. The same operation occurs for the sound relational data.The sound filename, '/n/virgo/work/mdbms/snd/jeff', goes in the f_id of relation VOICE4. The user gives the description of the voice as "slow voice" which is inserted. Once inserted, the data remains in the database management system until it is deleted. Other tuples are inserted in the same manner. An understanding of the retrieval process must occur first to understand the deletion operation. Examples of the retrieval operation and MDBMS deletion processing are now presented.

1. Retrieval of Information for a User Query

Simple queries designed and implemented by Pongsuwan [Ref. 15], are reviewed since it is the first step of deletion. I built the deletion operation on the existing retrieval process created by him. After completing the deletion and modification operations, these were joined with the work of my companion and can now process complex query modification and deletion. Retrieval by the user issuing query conditions to the MDBMS is accomplished via the main menu. The MDBMS retrieves the user data requirements as specified and places them into temporary structures that are mentioned in Chapter II. At this point, the user is shown the data to ensure that the data retrieved meets the criteria the user stated. Each set of formatted data and the attached image or sound media data, as applicable, is displayed tuple by tuple. The user is asked if he wants to see the media data of the parent relation. The tuples will be displayed in the order they are retrieved from the database. If the user wants to see the photo or hear the voice from the fourth tuple meeting the conditions, he is not able to alter the previewing order in the MDBMS prototype configuration Pei designed [Ref. 14]. In the current implementation for complex query processing Aygun [Ref. 2], the user is given a choice as to the order of the tuple media picture data he desires to see first. He will also have the option of which voice he wants to hear. This more user friendly style allowing the user to choose is the contribution of Aygun [Ref. 2] to the prototype. Also in [Ref. 2], additional operations such as nested queries and aggregate functions - in, max, in, intersection, and union - have been added to the prototype.

Understanding the temporary table functions was the first step for construction of the deletion and modification operations that include media. First, I had to learn what INGRES precompiled code results are produced when user query conditions are expressed. To work with precompiled code, I logged directly into INGRES (shown in the Appendix) and created small database operations for INGRES to manipulate. Some of the INGRES precompiled commands are in Chapter V. As mentioned, the user must retrieve the tuples from the tables first in order to remove the items based on user conditions. To illustrate this, we will query the database for a formatted example of retrieval and then retrieve data including media attributes. In the first query, we want to see the name and rank of the officers who reported in 1990 from the OFFICER relation. In the second query, we want to view the officer's pictures, hear their voice and see their names from the OFFICER relation when the officer's reported in 1990 and has the voice description "slow voice".

Relation name OFFICER

o_id	o_name	rank	salary	rep_yr	photo	voice
100	Jeff Kulp	Capt	10,000.00	1990	1	1
101	Fred Pong	Cdr	8500.00	1988	2	3
102	Vince Lum	LCDR	8000.00	1989	3	4
103	Mary Green	LTJG	7200.00	1991	4	5

Relation name PHOTO4

i_id	f_id	descrp	height	width	depth
1	/n/virgo/mdbms/gif/jeff.ras	big head	640	480	8
2	/n/virgo/mdbms/gif/fred.ras	green eyes	640	480	8
3	/n/virgo/mdbms/gif/plum.ras	small nose	640	480	8
4	/n/virgo/mdbms/gif/mary.ras	brown hair	640	480	8

Relation name VOICE4

s_id	f_id	descrp	freq	sampl	resol	encod
1	/n/virgo/mdbms/snd/jeff	slow voice	10	10	10	4
3	/n/virgo/mdbms/gif/fred	high voice	10	10	10	4
4	/n/virgo/mdbms/snd/vince	strong voice	10	10	10	4
5	/n/virgo/mdbms/snd/mary	weak voice	10	10	10	4

Figure 4.3 Media Relation Schemes for Navy Ship Media Attributes

Since the retrieval operations have been summarized in Chapter II, I will concentrate on the actions surrounding the temporary tables and the removal of the data from the MDBMS database.

a. Formatted Data Retrieval Operation

Examples to illustrate retrieval of formatted data are shown below.

Formatted QUERY 1:

What are the officers names and ranks that reported in the year 1990?

The SQL equivalent statement for this query is written as follows:

SELECT: o_name, rank (*which attributes are wanted*)

FROM: table (*which relation table is the information coming from*)

WHERE: officer.rep_yr = 1990 (condition)

Because this query contains only formatted data, it is passed directly to INGRES to retrieve the result. The result is placed in a temporary table away from the stored data and then displayed for the user to review.

Another example of formatted data retrieval:

Formatted QUERY 2:

What is the captain's name on the U.S.S. Elliott?

The SQL statement for this query is written as follows:

SELECT: o_name

FROM: ship,officer(*two tables are necessary to process this query*)

WHERE: ship.s_name = "Elliott" and ship.capt_id = officer.o_id.

Again, this is an example of a query that contains only formatted data. The query is passed directly to INGRES to get the result.

b. Media Data Retrieval Operations

When media data is included in the query, the query is decomposed into two parts as mentioned in Chapter II. The first sub-part is of the formatted data and is sent directly to INGRES for processing as shown in the examples above. The media data sub-part must be handled separately and the two sub-part results reconstituted into the final result. The user only sees the recomposed final result. Sample queries with varied holdings are shown below.

Media QUERY 1:

What are the names of the officers to see their photos and hear their voices if they reported in 1990 and have "slow voice". Show the officer photos that meet the conditions and play the voices.

The extended SQL statement for the query is written as follows:

SELECT: o_name, picture, voice

FROM: officer

WHERE: officer.rep_yr = 1990 and officer.voice(CONTAINS, "slow voice");

The decomposition of the query is listed below:

CREATE TABLE T1 as: *(temporary formatted data table)*

SELECT * *(all)*

FROM officer

WHERE officer.rep_yr =1990;

CREATE TABLE M1 as: *(temporary media data table)*

SELECT s_id

FROM VOICE4

WHERE VOICE4(CONTAINS, "slow voice");

CREATE TABLE RESULT as:

SELECT o_name, picture, voice

FROM T1, M1

WHERE T1.photo = M1.i_id

The final result is in an INGRES relation. This result of the officer names, the officer photo and hearing the officer voice are what will be seen by the user.

Media QUERY 2:

Find all ship weapon names where the ship has "gas turbine engine" in the photo description.

The extended SQL statement for the query is written below:

SELECT: w_name

FROM: ship, ship_weapon

WHERE: ship.photo(CONTAINS, "gas turbine engine") and ship_weapon.s_no = ship.s_no.

The decomposition of the query into sub-parts is listed here:

CREATE TABLE T1 as: (temporary formatted data table)

SELECT *

FROM: ship, ship_weapon

WHERE: ship_weapon.s_no = ship.s_no.;

CREATE TABLE M1 as: (temporary media data table)

SELECT i_id

FROM: ship

WHERE: PHOTO1(CONTAINS, "gas turbine engine");

CREATE TABLE RESULT as:

SELECT: w_name, photo

FROM: T1, M1

WHERE: T1.photo = M1.i_id

The INGRES operation of *cursor_output* is executed to process these queries. Cursor_output retrieves the result table data one tuple at a time and displays it to the screen. If media data is part of the RESULT table, then the tuple corresponding to the i_id and or s_id in the result table are retrieved from the media relations. The media data is displayed or played following the corresponding formatted data. The cursor_output format is shown below. See Figure 4.4. The cursor is initiated. The system retrieves everything from the result table and then retrieves the media identifiers (i_id or s_id) from the remitted data table in order to retrieve the media data.

```
EXEC SQL CREATE CURSOR cursor_output AS
SELECT * (all)
FROM RESULT
EXEC SQL FETCH CURSOR cursor_output;
print formatted data;
EXEC SQL CREATE CURSOR cursor_output AS
SELECT media data
FROM RESULT
EXEC SQL FETCH CURSOR cursor_output
display pictures;
play voice recordings; (not applicable for this example)
```

Figure 4.4 INGRES Pre-Compiled SQL Code

2. Approach to Deletion of Data from the MDBMS

As outlined in the previous section, the retrieval of the data to be deleted is first gathered based on the user specified query and the result is placed in a temporary table. The data is then displayed to the user before the temporary table of information is destroyed. What the user does not see going on in the delete operations is the matching of the database data to the temporary table data. When a tuple in a relation is to be deleted, not only the tuple in the relation must be deleted, but also the tuples in the media relations, if any, must be deleted as well. The SQL commands to delete from the actual relational data tables where the stored data matches the temporary result must be generated. The process of removing the temporary table is the last step in the deletion operation. INGRES embedded SQL calls this process "dropping" the table.

Initially to get the deletion operation to work in the MDBMS, a switch statement was used in the retrieve module. The retrieve module does half the work of the delete operation as it recalls the information the user no longer wants to store in the MDBMS.

After retrieval, the switch to deletion of this retrieved data, incorporated in the main retrieve process, completed the second part of the process. This technique of reusing portions of the retrieval code is an example of modularization and its benefits. The switch permits the retrieval of data to take place during execution of the retrieve operations as indicated from the main menu in *db*. It allows the execution of the deletion procedure when that mode is indicated by the user selecting deletion from the main menu. As seen in the code, execution of the retrieval operation occurs first. The results are displayed to the user. The temporary result table that stores the formatted data is then compared with the actual data stored and once the match is made, the media data is located in the actual storage area and tagged. Once these matching operations are complete, media data is deleted from the original table. The formatted data is then removed from the actual database and the temporary table is removed.

a. Main procedures for the Delete Operation

The actual operations for the deletion procedure is selected from the main menu are as follows:

retrieve (DEL_MODE): The main retrieval module is executed to retrieve the user's query conditions. The mode statement flag, DEL_MODE, is passed in the functions for execution of the eventual deletion of data.

table_cursor = table_entry(): This sets the pointer to the table where data will be deleted.

get_all_atts_of_a_given_table(): This function retrieves all the attributes of the tuples that meet the user specified conditions.

ql_retrieve(RTRVE_MODE):The retrieval process occurs in this function. Here other procedures are called to locate the media data if it is involved in the query.

ql_print_delete_data():This procedure matches the data stored in the database to the data in the temporary table. Then other sub routines such as *get_rid_image* and *get_rid_sound*, are called to remove the media data from the database.

ql_retrieve(mode): Using the delete mode this time, this procedure will execute the deletion of the formatted data that matched the temporary table conditions the user wanted removed.

drop_table(temp_table): INGRES command to remove the temporary table created to hold the to-be-deleted data.

The ql_print_delete_data is the procedure where the comparison is made. This procedure in turn calls the procedure get_rid_image or get_rid_sound if media data is included. If not, then these operations are bypassed and the retrieve operation is called again. This time, the delete mode to delete the data stored in the actual table invoked. The temporary table is "dropped" as specified in INGRES SQL fashion. I was able to make the switch statement work once I had an understanding of how to modify the SQL commands.

In Appendix A, the Retrieve module shows the code with implementation of the deletion operations incorporated with an "if" statement where a switch statement was initially tested. Each of the INGRES precompiled commands within the procedures are differentiated by their synchronization characters. For the INGRES precompiled command of retrieve, the synchronization is "0". For delete, the synchronization character is "1", for insert the character is "3,"and so forth. A way to identify precompiled commands is by locating commands beginning with the "II". All INGRES embedded SQL precompiled commands have the ""II" characters. Examples are ""IIwritedb("=") and "IIcsrQuery((char *) 0)". More precompiled code examples will be in the A and B Appendices.

b. Testing Methodology

Once the switch mode was integrated into the retrieve module code, trial executions of a simplified user conditions delete began. The first runs dealt with only one tuple of strictly formatted data. When several variations of the one formatted data tuple worked, we then tried the next level of increased media data processing difficulty. The query was modified such that more than one tuple would meet the user criteria. Another pointer to loop through the database looking for more than one tuple matching the user query was required. As the pointer looped through to find matches to the query, the accountability of the cursors and pointers gained attention. Several counters to maintain record of the pointers pointing to data stored in the INGRES database management and counters for the cursors to the data in the temporary tables were required.

Once the delete operation for multiple formatted tuples was working, then the obstacle of the one tuple with one media type was next. The operation to delete one tuple of formatted data with one media type image is as follows: first the same retrieval operations had to occur to get the table name and conditions from the user. Once the desired table was retrieved, a check for media data was performed. If media data is included, then the relation table key is found and the media relation table is located by using the table key and the media i_id for image or s_id for sound data. The media data that meets the specific query is deleted. After the media data is removed, then the formatted data is then deleted. This process is repeated and extended to handle an increasing number of tuples possessing one media attribute. When the formatted tuple with one media works for either voice or image, testing returns to just one tuple containing two media attributes. Successful two media data in one tuple test leads to creating another looping mechanism for more than one tuple with two media attributes. This configuration is similar to the one used for the multiple tuple formatted data case. Media tables stored as abstract data types are handled in such a way that the user is unaware of the separate tables accessed when deletion is called.

C. USER INTERFACE EXAMPLE

When the user wants to delete some information already residing in the database, the user must first select the "Delete" option on the MDBMS main menu. The database will prompt the user with line by line commands and instructions for manipulating the data in the database. These instructions will continue to process the user's query step by step as appropriate responses are provided by the user to each database operation. The retrieval and deletion of the first query example will be processed as shown on the screen of the MDBMS. User responses are shown in the cursor encased (<>) italic bold type. The initial screen below shows the main menu of the MDBMS prototype.

```
Multimedia Database Management System
=====================================
1. Create Table
2. Insert Tuple
3. Retrieve
4. Delete
5. Modify
6. Print out current data information (test purpose)
0. Quit
=====================================
Select your choice: <4>
```

Here is the main menu from the MDBMS that is shown on the screen. The user must select the Delete operation and then hit a carriage return <cr> to continue the operation. The following is the response to the formatted data only query of "*List the ships in the database of type 'carrier'?*"

Your Selection is Delete!

1. Retrieval Conditions - Input Phase

The user is requested to enter a temporary table name to store the results of the query while it is being processed by the MDBMS. After providing the temporary table name, the user is asked in which table inside the MDBMS he wishes to remove data. The conditions, if they exist, for specific attributes in the chosen table are next requested from the user. If there are more conditions that are expected to be boolean "*and*" together, then the group condition query needs a <"Y" > from the user.

Hit return to continue (any other key to QUIT)

Enter a table name to hold the temporary result of the query: <***demo_table***>

Select the table(s) separated by commas <,>:(<?> for HELP!)

SELECT TABLE(S): <***ship***>

Any condition? (Y/N): <***Y***>

Group Condition? (Y/N): <***N***>

Enter the attribute name: <***type***>

Enter the Condition: <= “***carrier***”>

2. Execution Phase

The checks of a repetitive table name and if the table and attributes given by the user already exist in the database are performed. When all the data provided is correct, then no error statements occur. If the table is not correct or the data given for the attributes does not exist, an error message is returned to the screen. The following was executed as the table and attribute critierias were found.

Searching.....

WHERE = “carrier”

wait...

There are 2 records that match the delete query:

record id 1 name: Elliott

record id 2 name: Ticonderoga

The following photo has been found:

Number: 1

Description:

<<large ship with many weapons.>>

Record no1 filename: /tmp_mnt/n/virgo/work/mdbms/MDBMS/ 91133.19734

Do you want to see the photo? (Y/N): ***Y***

Show image...

****(Photo is displayed on the screen if user types in Y)**

Record no 2 filename: /tmp_mnt/n/virgo/work/mdbms/MDBMS/ 90206.30421

Do you want to see the photo? (Y/N): ***Y***

Show image...

****(Photo is displayed on the screen if user types in Y)**

Do you want to continue to Delete? (Y/N): ***Y***

media data matching the query is being deleted from ****picture1*****

data is being deleted from ******SHIP*****

(return to the main menu) **

As stated, the above query execution handled only formatted data, the next example has the media data included in the query.

The above is what the user sees on the screen for the query of ships of type carrier. The following query example is the illustration of the operation from an internal view.

1. CREATE TABLE F1 AS

SELECT s_name, photo

FROM SHIP

WHERE ship.type = "carrier"

2. CREATE TABLE M1 AS

SELECT i_id

FROM PHOTO1

WHERE (F1.photo = M1.i_id)

3. CREATE TABLE RESULT AS

SELECT all

FROM F1,M1

4. MATCH TABLE RESULT AS

SELECT all

FROM SHIP, PHOTO1

WHERE ship.attribute = result.attribute AND photo1.i_id = result.i_id

5.DELETE TABLE PHOTO1

WHERE RESULT.i_id = PHOTO1.i_id

(all items in the result table are those media values meeting the user conditions)

5.DELETE TABLE SHIP

WHERE RESULT.attribute = SHIP.attribute

(the items here are all the formatted values in the tuple that meet the user conditions)

6. DROP TABLE RESULT

(removes temporary table)

Query example 2:

This example will show how the MDBMS handles the media data. The sub-parts remain transparent to the user and the final result is displayed for the user prior to the deletion procedure. More than one table may be accessed to get the relations for the user. The functions and procedures that make the delete operation work are outlined in the previous section. The steps are the same as in the first example except for the handling of the media portion of the query. The actual media operations are not shown. The query is "*Display the officer of the ship USS Elliott who has a photo description* ***'He has big head'***?".

Multimedia Database Management System

==================================

1. **Create Table**
2. **Insert Tuple**
3. **Retrieve**
4. **Delete**
5. **Modify**
6. **Print out current data information** (test purpose)

0. **Quit**

==================================

Select your choice: <4>

Here is the main menu from the MDBMS that is shown on the screen. The user must select the Delete operation and then hit a carriage return <cr> to continue the operation. The following is the response:

Your Selection is Delete!

Enter the table name to hold the temporary result of the query: ***<demo_table>***

Select the table(s) separated by commas <,>:(<?> for HELP!)

> SELECT TABLE(S): ***<officer, ship>***
>
> Select the attribute(s) separated by comma <,>:(<?> for HELP!)
>
> (Hit <ESC> for no attributes) ***<o_name>***
>
> Any condition? (Y/N): ***<Y>***
>
> Group Condition? (Y/N): ***<N>***
>
> Enter table name: ***<ship>***
>
> Enter attribute(s): ***<name>***
>
> Enter Condition: ***<= "Elliott">***
>
> Enter table name: ***<officer>***
>
> Enter attribute(s): ***<o_id>***

Enter Condition: < ***o_id = ship.capt_id***>

Enter table name: <***officer***>

Enter attribute(s): <***photo***>

Please enter your query description

*noun phrases separated by commas and end with an exclamation mark

*sentence end with a period.

(end whole description with an empty line);

he has big head.

Searching....

There are 1 records that meet the delete query.

record id 1 o_name: Jeff Kulp

Record no 1 filename:/tmp_mnt/n/virgo/work/mdbms/MDBMS/91133.19734

Show image...

The following photo has been found:

Number: 1

Description:

<<he has big head>>

Do you want to see the photo? (Y/N): ***Y***

(Photo is displayed on the screen)

sound management

Record no 1

Play the sound? (Y/N): ***Y***

(Sound is played for the user)

Do you want to continue to delete? (Y/N): ***Y***

media data matching the query is being deleted from ****photo4*****

media data matching the query is being deleted from ****voice4*****

data is being deleted from ******OFFICER*****

(return to the main menu) **

If the user desires to see and hear the media data types that will be removed, the user may display the photos and play the sound. The main menu is then re-displayed with the basic database management options as the deletion operation occurs in the background. The user can then re-display the table he just altered to verify the operation deleted the data as planned. The above is what the user sees on the screen for the query of ships of type carrier. The following query example is the illustration of the operation from an internal view. Notice that more temporary tables are created as the query becomes more complex.

1. CREATE TABLE F1 AS

SELECT capt_id

FROM SHIP

WHERE ship.name = "Elliott"

2.CREATE TABLE F2 AS

SELECT all

FROM OFFICER

WHERE o_id = shF1.capt_id

3.CREATE TABLE M1 AS

SELECT i_id

FROM PHOTO4

WHERE CONTAIN(PHOTO4, "he has big head")

4CREATE TABLE RESULT AS

SELECT all

FROM F1,F2,M1

WHERE (F1.capt_id = F2.o_id) and (F2.photo = M1.i_id)

5. MATCH TABLE RESULT AS

SELECT all

FROM OFFICER, PHOTO4

WHERE officer.attribute = result.attribute AND photo4.i_id = result.i_id

6.DELETE TABLE PHOTO4

WHERE RESULT.i_id = PHOTO4.i_id

(all items in the result table are those media values meeting the user conditions)

7.DELETE TABLE SHIP

WHERE RESULT.attribute = SHIP.attribute

(the items here are all the formatted values in the tuple that meet the user conditions)

8. DROP TABLE RESULT

(removes temporary table)

The deletion procedure is the second in the three phases of modification. Retrieval being the first, deletion the second and the execution of the actual modification the third step in the process. Chapter V will cover the modification procedures beyond what has been presented in these first four chapters.

V. MODIFICATION METHODOLOGY

Solving the modification problem began with designing and/or enhancing functions and procedures that are compatible with the currently working modules and procedures in the MDBMS system. The modularization process discussed in Chapter III of this thesis was the first step. The MDBMS prototype left by Pei and Atila when they completed their work provided ample room for improvements and completion of the system. Aygun and I commenced work on the completion of the MDBMS areas not currently implemented and improved some areas already in the running prototype. My design ideas were to ι data structures used in current management systems for implementation of the retrieval module and to design additional data structures necessary to capture the user information required for processing and modifying formatted and media data, such as image and sound, if they existed. I made use of the existing implementations, whenever appropriate, for some of the operations to update the data. Extension of the user interface in the existing MDBMS allows for modification of the various data already stored in the MDBMS.

A. INTRODUCTION

Conventional database management systems have modified standard data for some time. The modifying media data types in the MDBMS provide many researchers the opportunity to discover the ideal method. In the MDBMS prototype, the media data is not placed in a temporary storage structure as the standard data is stored. Media data types are tabbed in their original storage table. If a tuple that possesses a media type is modified, the new media portion is added to the appropriate media table, be it photo or sound. After the newly modified data has been entered, then the s_id and i_id of the original tuple data is removed from the parent table. The data actually is a new tuple relation in the database in regards to the media portion even if the media data is not altered. This technique is designed based on the original structures that Pei created in his insertion operations. The internal catalog system creates new numbers for each new photo imputed.

1. Why We Need a Modification Option

The modification steps are needed because most users are human and make errors when entering data. Another reason is to have the capability to update and modify the existing data is the data continually changes. The fleet database is a living document that continually changes as the area of the home base for many ships and military installations close with force reductions, captains are transferred or retire, ship numbers change as force realignments occur, and newer weapons systems are retrofitted on older ships. These are only a few occurrences of why military business applications and complete database management systems require modification options to the data after it has been stored.

2. How modification works in general

The steps used in the modification construction process are first obtain the data using the retrieval operations described in previous chapters, and second, to use similar modification procedures as those Pei designed to modify data in the insert operation. This way, the re-thinking of pre-compiled INGRES code is minimized. The media data presents some unique problems as the catalog management system generates sequential numbers for each attribute of every tuple when they are first inserted using Pei's functions. Finding the proper way to alter the Pei data structures or to create similar ones that are compatible with the other Pei designed modification procedures became the focus of the operation of modifying already stored data.

3. Design Issues for the Information Storage in the MDBMS

In the prototype construction of the update operation, [Ref. 14] there exists a modification call for data that has not been inserted into the database yet. This data has not been placed in the INGRES system but is stored in temporary arrays. Pei arranged to query the user for changes before the data becomes stored in the INGRES tables. The difference in a modification operation on data already stored in INGRES and data just being added to the database lies in the handling of the data in the temporary arrays. See Figure 5.1. This data was corrected and then inserted, one data type at a time, in the sequential and numerical order as defined in the relational catalog, one tuple at a time. Retrieval of the data for modifications is not a problem. Once the user made the corrections on the data, the data was inserted. Inserting the data in a numerical order is not possible when the modification is of data already stored in the MDBMS. Therefore, maintaining the knowledge of where the pointers to the information are, becomes very important. The storage of the values in respective value array structures proved to be a way to keep track of the information the user was attempting to modify. Another concern was where to place the data upon completion of the modification operation so that it does not cause storage addressing problems when retrieving the data at a later date. User's do not always want to change everything in a tuple or in a specific manner. Pei uses an index called the active_media_list to maintain accountability of the insertion of media data types. His method cannot be applied to the modification of data inside the database, as INGRES generates numbers internally that would confuse the indexing mechanism.[Ref. 14, Ref. 15] For modification of data already stored inside the database, media flags are used in the modifying of stored data. The media data flags alert the MDBMS that media data is involved in the processing of the data values the user wants to modify. The media flags, as shown in the prototype program, let the counters know when they must be incremented as the media data is processed. If the changes do not occur to the media data, then the same file_id is provided to the media data in the new tuple relationship.

Table List

1
2
3
4

Table Array

TableName	table_key	att_count	attr_entry
ship	1	10	1
ship_weapon	2	2	11
weapon	3	5	13
officer	4	7	18

	Int_value	F_value	C_value	IMG_value	SND_value
1	100		Jeff Kulp	1	1
2	10000		Capt	2	3
3	1990		Fred Pong	3	4
4	101		Cdr	4	5
5	8500		Vince Lum		

Figure 5.1 Table List, Table Array and Value Array Tables for the Catalog Using the Officer Table as the Data Entries

att_name	data_type	media_id	next_index	value_entry
18 o_id	integer	-1	19	0
19 o_name	character20	-1	20	0
20 rank	integer	-1	21	1
21 salary	integer	-1	22	2
22 rep_year	integer	-1	23	3
23 photo	image	4	-1	0
24 voice	sound	4	-1	0
25 misn_id	integer	-1	26	0

Figure 5.2 Att_Array for the Catalog Using Officer Table Data Entries

B. DESIGN AND IMPLEMENTATION FOR MODIFICATION

1. Testing Methodology

Once the switch mode structure was integrated into the retrieve module code, trial executions of a simplified data modification began. The first runs dealt with modifying one attribute of formatted data in only one tuple. When several attempts to modify one attribute of formatted data, either integer, character, or float, for one tuple worked, we tried modifying more than one attribute of formatted data for a single tuple. This did not present any problems as the values of the different standard values are each stored in their own value arrays as mentioned in Chapter II. See Figures 5.1 and 5.2. Since each array contains the respective value type, the index to the entry in the value array is entered into the value_entry column of the Att_array for that attribute. These arrays and the media flags that are introduced later provided a means of proper manipulation of the internally stored data. After one tuple was able to be modified, then new queries that would give more than one tuple as a result were tried. This multi-solution method required another pointer to loop through the database looking for more than one tuple. As the pointer looped through to find matches to the query, the accountability of the cursors and the pointers gained attention. Several counters and boolean flags to maintain records of the pointers to data stored in the INGRES database management and the data in the temporary tables were added.

When the delete operation for multiple formatted tuples was working fine, the obstacle of the one tuple with one media type in the relation was the next test of the system. The operations to delete one tuple of formatted data with one media type image are as follows: First the same retrieval operations to get the table name and conditions from the user must occur. Once the desired table is retrieved, a check for media data is performed. If media data is included, then the relation table key is found and the media relation table is located by using the table key and the media i_id for image or s_id for sound data. The media data that meets the specific query is deleted. After the media data is removed, the formatted data is deleted. This process is repeated and extended to handle an increasing number of tuples possessing one media attribute. When the formatted tuple with one media works for either voice or image, then testing returns to just one tuple containing two media attributes. Successful two media data in one tuple test leads to creating another looping mechanism for more than one tuple with two media attributes. This configuration is similar to the one used for the multiple tuple formatted data case. Media tables stored as abstract data types are handled in such a way that the user is unaware of the separate tables accessed when deletion is called. In the modification process, however, the user participates in the processing of the media tables to a limited extent because media relation attributes, such as the media data's f_id and description, can be modified.

After this modification code worked with the simple retrieve operations created by Pongsuwan, it was integrated into Aygun's complex query version of retrieve. The merging of the retrieval, modification and deletion modules forced a change in the programming code structures each of us developed separately. As we worked together, the switch statement that I mastered for modification and deletion, would not work properly in the integrated code. The switch statement overlapped a user choice retrieval menu that queried for complex queries. An *if* statement was substituted for the switch statement to achieve a working composite program.

2. Main Procedure for the Modify Operation

Initially, several flags are set to maintain a count on the formatted, image, and sound types of data in the MDBMS. The actual operations for the modify procedure after choice five is selected from the main menu are as follows:

retrieve (RTRVE_MODE): The main retrieval module is executed to retrieve the user's query conditions. The mode statement of RTRVE_MODE is passed first to locate the data that meets the query. MOD_MODE variable is passed to the main retrieve function so that the actual execution of the modification on the retrieved data takes place.

table_cursor = table_entry(): This sets the pointer to the table where data will be matched to the data stored that eventually will be modified and re-inserted.

get_all_atts_of_a_given_table(): This function retrieves all the attributes of the tuples that meet the user specified conditions.The user will then have the tuple available to modify all attributes before re-insertion

ql_retrieve(MOD_MODE): The modification processes are executed when the modify mode in the if statement occurs in this function call. Procedures are called to locate the media data if it is involved in the modification query.

flags set to false for formatted, image and sound data: These flags help maintain the counts on the indices to the tuples to modify.

print_for_modify(c): This procedure is embedded INGRES code that retrieves the tuple values that meet the conditions the user states and prints out the tuples so that the user may see how many and which tuples are to be modified. The user will be asked to modify each tuple one by one in the next procedure.

process_tuple_by_tuple(r): This procedure is the function that checks for multiple tuples meeting the user query criteria.When more than one tuple needs modification, the first in the database called will be modified first and continue to loop through the database to search for other tuples. After all those that the user requested are modified, this process terminates and each of the tuple changes is shown to the user in the procedure below.

mod_display_tuple(mode, media_counter): This procedure shows the user the tuple information with the changes he has made to the original tuple. This procedure in turn invokes ***mod_print_tuple*** and ***mod_print_media_tuple***. If more modification is necessary for the tuple after the user sees it, the user is able to make those corrections during the continued execution of this procedure.

store_data(mode): This procedure is invoked every time the catalog information in the system tables have been updated or modified. The procedure further performs processes from the system tables in the main memory to host three catalog files in external storage devices.

mod_ql_insert_tuple(mode): During modification, this procedure generates the user's SQL statements for data modification. It constructs the SQL commands for the insertion of a modified tuple. The sub-procedure of the *mod_ql_insert_tuple* is called to generate the user SQL statements for the data insertion of the modified media relations. The SQL commands create the media relations and the INGRES.

delete_for_modify(r): This procedure will execute the deletion of the media data that the user wanted modified. This procedure matches the data stored in the database to the data in the temporary table. Then other subroutines are called, such as *mod_get_rid_image* and *mod_get_rid_sound*, to remove the media data from the database.

delete_formatted_part_for_modify(): This procedure will execute the deletion of the formatted data that the user wanted modified.

drop_table(temp_table): INGRES command to remove all the temporary tables created to hold the data while processing the modification operation.

reset media flags.

C. USER INTERFACE EXAMPLES

For modification in the MDBMS, the user must follow the modify procedures as outlined in the main menu. As expected, the user must first state the conditions of the modification. The data is retrieved and then tuple by tuple, the user may modify any and all of the attribute values currently stored in the MDBMS. The prototype system prompts the user for the data necessary to process the user request. The actual user responses in this example are encased in the cursor (<>) in bold type. Two examples are given, one dealing with formatted data and the other containing some image data.

1. Query 1 - Formatted Data Modification

The first query to modify is *"Find the officer with the name Fred Pong and change his rank to CAPT and his salary to $8700."* In this query, there is not a need to connect to the parser to analyze a media description. The original table information meeting the user conditions will be displayed to ensure it is the correct relation to be modified. After displaying the information, the user will respond to continue to modify or terminate the current operation. When continuing the modification operation, the user is asked which items need modification. The retrieve and deletion operations will not be explained in depth as they are stated in previous chapters. Refer to Figure 4.3 to see the values of the relations and Figure 5.3 to see the values upon completion of the modification operation.

MULTIMEDIA DATABASE MANAGEMENT SYSTEM
====================================
1. **Create Table**
2. **Insert Tuple**
3. **Retrieve**
4. **Delete**
5. **Modify**
6. **Print out current data information**(test purpose)
0. **Quit**

Select your choice: <5>

Your selection is **MODIFY**!

Hit Return to continue!(Any other key to QUIT!)

Enter a table name to hold the temporary result of the query:< **was**>

Select the table(s) separated by a comma <,>: (<?> for HELP!)

SELECT TABLE(S): <***officer***>

Any condition? (y/n): <***Y***>

Group condition? (y/n) <***N***>

Enter attribute name: <***o_name***>

Enter the condition

= "***Fred Pong***"

Searching...

Where = "Fred Pong"

There are 1 record(s) that match the query

Press ENTER to continue...

You have 1 tuples to modify

There are 1 tuples to be processed!

Press ENTER to continue...

Table Name: officer

Order Attribute Name Data Type Value

1o_id integer 100

2 o_name c20 "Fred Pong"

3 rank c20Cdr

4 salary integer 8500.00

5 rep_yr integer 1988

6 photo image HAS VALUE

7voice sound HAS VALUE

8 description '/work/mdbms/MDBMS/91216.194349',

'he has green eyes.',

'/work/mdbms/MDBMS/91216.194350',

'he has high voice.',

Any change before insert? (y/n):<*Y*>

Select the order which you want to change its value:

Any other key to cancel the operation! Select:*<3>*

Table Name: officer

Att_Name: rank

Data Type: c20

Value: Cdr

Please Enter <<c20>> Value (? if unknown): ***Capt***

Table Name: officer

Att_Name: rank

Data Type: c20

Value: 'Capt'

Any More Change? (y/n): ***y***

Table Name: officer

Order Attribute Name Data Type Value

1o_id integer 100

2 o_name c20 "Fred Pong"

3 rank c20Capt

4 salary integer 8500.00

5 rep_yr integer 1988

6 photo image HAS VALUE

7voice sound HAS VALUE

8 description '/work/mdbms/MDBMS/91216.194349',

'he has green eyes.',

'/work/mdbms/MDBMS/91216.194350',

'he has high voice.',

****

At this point, the user has changed one of the tuple values and is asked if more tuple values are to be modified. After this sequence is executed and all the values are modified, there is one last viewing of the tuple values. If the values are to the user's liking, then the old values are removed from the database and the temporary tables as shown in chapter IV.

****

Select the order which you want to change its value:

Any other key to cancel the operation! Select: ***5***

Table Name: Officer

Att_Name: salary

Data Type: integer

Value: '8500'

Please Enter <<c20>> Value (? if unknown):***8700?***

Table Name: Officer

Att_Name: salary

Data Type: integer

Value: '8700'

Any More Change? (y/n): ***N***

Table Name: Officer

Order Attribute Name Data Type Value

1o_id integer 101

2 o_name c20 "Fred Pong"

3 rank c20Capt

4 salary integer 8700.00

5 rep_yr integer 1988

6 photo image HAS VALUE

7voice sound HAS VALUE

8 description '/work/mdbms/MDBMS/91216.194349',

'he has green eyes.',

'/work/mdbms/MDBMS/91216.194350',

'he has high voice.',

Select the order which you want to change its value:

Any other key to cancel the operation! Select: <cr>

****

The user has completed the data modification process based on the formatted data query. Now the INGRES insertion process takes place and then the deletion of the original data and temporary tables occurs.

****

SQL statement:

insert into officer (o_id,

o_name,

rank,

salary,

rep_yr,

photo,

voice)

values ('101',

'Fred Pong',

'Capt',

8700,

19988,

```
2,
3);

INSERTING STD TUPLE NOW. PLEASE WAIT!
INSERT A STD TUPLE COMPLETE!
press ENTER to continue
insert into photo4 (i_id,
 f_id,
 descrp,
 height,
 width,
 depth)
 values(3,
 '/work/mdbms/MDBMS/91216.194349',
 'he has green eyes.',
480,
 640,
 8);
INSERTING MEDIA TUPLE NOW. PLEASE WAIT!

INSERT AN IMAGE TUPLE COMPLETE!
press ENTER to continue
insert into voice4(s_id,
 f_id,
 descrp,
frequency,
sample,
resolution,
```

encoding)

values(4,

'/work/mdbms/MDBMS/91216.194350',

'he has high voice.',

10,

10,

10,

4);

INSERTING MEDIA TUPLE NOW. PLEASE WAIT!

INSERT A SOUND TUPLE COMPLETE!

press ENTER to continue

(return to main menu)

2. Query2 - Media Data Modification

This query involves the user specifying the conditions based upon the description of the voice. After finding the data entries that match the user's conditions, modification operations for this data occurs. The query "*locate the officer with a voice description of 'strong voice' and change it to say 'loud voice' then check his photo and change the description to state 'wears glasses' if he has them in the photo*".

MULTIMEDIA DATABASE MANAGEMENT SYSTEM

====================================

1. **Create Table**
2. **Insert Tuple**
3. **Retrieve**
4. **Delete**
5. **Modify**
6. **Print out current data information**(test purpose)
0. **Quit**

Select your choice: <**5**>

Your selection is **MODIFY**!

Hit Return to continue!(Any other key to QUIT!)

Enter a table name to hold the temporary result of the query:< ***met***>

Select the table(s) separated by a comma <,>: <?> for HELP!)

SELECT TABLE(S): <***officer***>

Any condition? (y/n): <***Y***>

Group condition? (y/n) <***N***>

Enter attribute name: <***sound***>

Please enter your query description

*noun phrases separated by commas and end with an exclamation mark

*sentences ending with a period.

(end whole description with an empty line);

he has strong voice.

Searching....

Where officer.sound

voice4 = "he has strong voice"

There are1 record(s) that match the query

record id 4 o_name: Vince Lum

Record no 1 filename:/tmp_mnt/n/virgo/work/mdbms/MDBMS/93243.14538

Sound Management...

Number: 1

Description:

<<he has strong voice>>

Do you want to play the sound? (Y/N): ***Y***

(voice is played for the user on the IBM PC)

show image...

Record no 1

Do you want to see the image? (Y/N): ***Y***

(Image is displayed for the user)

Searching...

Order Attribute Name Data Type Value

1 o_id integer 102,

2 o_name c20 'Vince Lum'

3 rank c20 'LCDR',

4 salary integer 8000,

5 rep_yr integer 1989,

6photo image HAS VALUE

7voice sound HAS VALUE

Select the order which you want to change its valueTable Name: officer

Att_Name: voice

Data Type: sound

Value:

==>File_name: '/tmp_mnt/n/virgo/work/mdbms/MDBMS/93422.47329'

==>Description: <<he has strong voice>>

Change SOUND file name? (y/n): ***N***

Play the SOUND before enter the description? (y/n): ***n***

Change SOUND description? (y/n): ***y***

Please enter your modified description

* noun phrases separate by commas and end with an exclamation mark

* sentences ending with a period.

<<'he has loud voice'>>

Press Enter to Continue;

Table name: Officer

Order Attribute Name Data Type Value

1 o_id integer 102,

2 o_name c20 'Vince Lum,'

3 rank integer 'LCDR,'

4 salary integer 8000

5 rep_yr integer 1989,

6 photo image HAS VALUE,

7 voice sound HAS VALUE,

8 description '/work/mdbms/MDBMS/93456.23186'

'he has small nose.'

'/tmp_mnt/n/virgo/work/mdbms/MDBMS/978654.23432'

'he has loud voice',

Any change before insert? (Y/N): ***<Y>***

Select the order which you want to change its value:

Any other key to cancel the operation! Select: 6

Table Name: officer

Att_Name: photo

Data Type: image

Value:

==>File_name:

<<‘he has small nose’>>

Please Enter <<image>> File Name!

NOTE: Enter The Full Path Name: (? if unknown)

/tmp_mnt/n/virgo/work/mdbms/MDBMS/98345.25563

Display the image before enter the description? (y/n): ***n***

Change IMAGE description? (y/n): ***y***

Please enter your query description

 * noun phrases separated by commas and end with an exclamation mark

 * sentences ending with a period.

(end whole description with an empty line):

<<***he wears glasses.***>>

Table name: Officer

Order Attribute Name Data Type Value

1 o_id integer 102,

2 o_name c20 ‘Vince Lum,’

3 rank integer ‘LCDR,’

4 salary integer 8000

5 rep_yr integer 1989,

6 photo image HAS VALUE,

7 voice sound HAS VALUE,

8 description ‘/work/mdbms/MDBMS/93456.23186’

‘he wears glasses.’

‘/tmp_mnt/n/virgo/work/mdbms/MDBMS/978654.23432’ 'he has loud voice',

Any change before insert? (Y/N): <N>

f_id==>/work/mdbms/MDBMS/91216.194349

descrp==>he has loud voice.

file_id==>/work/mdbms/MDBMS/9678456.3342

descrp==>he wears glasses.

Connect to PARSER, Please Wait.....

Hit RETURN to Continue!

SQL statement:

insert into officer (o_id,

o_name,

rank,

salary,

rep_yr,

photo,

voice)

values (102,

'Vince Lum',

'LCDR',

8000,

1989,

5,

6);

```
INSERTING STD TUPLE NOW. PLEASE WAIT!

INSERT A STD TUPLE COMPLETE!
press ENTER to continue
insert into voice4 (s_id,
 f_id,
 descrp,
 freq,
 sample,
 resolution,
encoding)
 values(6,
 '/work/mdbms/MDBMS/91216.194349',
 'he has loud voice'.
',
 10,
 10,
 10,
4);

INSERTING MEDIA TUPLE NOW. PLEASE WAIT!

INSERT AN SOUND TUPLE COMPLETE!
insert into photo4 (i_id,
 f_id,
 descrp,
 height,
 width,
```

```
depth)
values(5,
'/tmp_mnt/n/virgo/work/mdbms/MDBMS/91216.194349',
'he wears glasses.',
480,
640,
8);
INSERTING MEDIA TUPLE NOW. PLEASE WAIT!

INSERT AN IMAGE TUPLE COMPLETE!
press ENTER to continue
***(return to main menu, modification complete!)******
```

These examples are reflected in Figure 5.3. The media tables for the tuples must be indexed as the change in the media data requires a new relation be added to the database.

Relation name OFFICER

o_id	o_name	rank	salary	rep_yr	photo	voice
100	Jeff Kulp	Capt	10,000.00	1990	1	1
101	Fred Pong	Capt	8700.00	1988	2	3
102	Vince Lum	LCDR	8000.00	1989	5	6
103	Mary Green	LTJG	7200.00	1991	4	5

Relation name PHOTO4

i_id	f_id	descrp	height	width	depth
1	/n/virgo/mdbms/gif/jeff.ras	big head	640	480	8
2	/n/virgo/mdbms/gif/fred.ras	green eyes	640	480	8
5	/n/virgo/mdbms/gif/plum.ras	he wears glasses	640	480	8
4	/n/virgo/mdbms/gif/mary.ras	brown hair	640	480	8

Relation name VOICE4

s_id	f_id	descrp	free	sampl	resol	encod
1	/n/virgo/mdbms/snd/jeff	slow voice	10	10	10	4
3	/n/virgo/mdbms/gif/fred	high voice	10	10	10	4
6	/n/virgo/mdbms/snd/vince	loud voice	10	10	10	4
5	/n/virgo/mdbms/snd/mary	weak voice	10	10	10	4

Figure 5.3 Media Relations After Modification of Data Values

VI. CONCLUSIONS AND RECOMMENDATIONS

A. REVIEW OF THESIS WORK

The NPS MDBMS prototype is a demonstration version of a Multimedia Database Management System which can manage formatted and multimedia data. With the completion of the deletion and modification procedures, the prototype now provides all the conventional database management operations. Items can be deleted or modified now after they are stored. Prior to the completion of this thesis, only the create, insert, and retrieve options were available. The system, as it is now, performs media data processing by means of executing the SQL commands step by step. These commands perform intermediate relations and sub query decompositions/recombination in order to attain the desired output. The final product, a relation containing the tuples is available to the user for review, update, delete, or display the data. Companion theses present the complex query retrieval [Ref. 2] and the ideal user interface for an MDBMS [Ref. 13] also incorporated in the prototype.

Repetitive updates can be made based on the descriptions of image, sound and formatted data can be changed or deleted permanently. The research done to define the deletion and modification operations, and specifically address data involving image and sound and the integration of media with the formatted data, is explained in the body of this thesis. This thesis concentrated on the complex methods and operations of data modification and deletion for the MDBMS. Solutions to some of the issues raised were found. The type of data structures required to capture the user supplied information for modifications on data and the coding for compilation into acceptable INGRES statements are two of these issues. With the completion of the deletion and modification procedures, the prototype, a complete database management system prototype able to manage different types of media data from creation to deletion, is obtained. This multimedia database management system demonstrates the great advantages users possess with a MDBMS over a traditional DBMS.

The prototype's structured foundation was established for the design and implementation of the functions that perform the user requested actions. These actions are implemented interactively with a line by line interface. The transformations of the user's query is entered into an embedded SQL request to be processed in INGRES and written in standard 'C' programming language. This query is replaced by embedded SQL compiled statements. Each corresponds to a set of function calls to operate on the appropriate standard and media sound or image - object relations.

B. FUTURE WORK

Future work areas overlap with those currently being researched as the MDBMS evolves into a better operating prototype. Research is currently going on for the appropriate graphical user interface for the MDBMS [Ref. 13]. This graphical interface will also incorporate a more detailed help tool which the user can easily operate the first time the he runs the MDBMS. The design of this user interface uses icons generously. Having better user help pull-down menus or help screens that provide definitions or examples of how the database steps are performed, would surpass the current line by line user interface that supports the MDBMS now. The graphical user interface will display more vividly the way the relations in the database are arranged and the transactions as they occur in the MDBMS. Further work envisioned is the implementation of this ideal graphical user interface designed in Peabody [Ref. 13]. An upgrade to an object oriented language will likely make a difference towards improving the user interface.

The principles of understandability, maintainability, correctness, reusability, and uniformity are team programmers goals in their products. The module hierarchy reflects the relationships and the way each module interacts with the others. More work can be done with the modularization of the MDBMS. Work completed by Aygun [Ref. 2], and myself was just enough to get the large program separated into some main modules. Due to a lack of time, more work to eliminate redundant code and consolidate similar operations was not accomplished. Future work in continuing our modularization efforts would make a better prototype.

Another idea for the improvement of the MDBMS is allowing the user to change a table format after it has been created and data inserted into the database. Currently, the data in a table that needs changing cannot retain the data as the table must be removed and then is restructured. The new option to alter the table and continue to use data already stored would be a more efficient way to manage data. An example of this is to alter a personnel table by adding a personal insurance attribute. To alter the personnel table attributes, once it is discovered that a column of data is needed, or is no longer needed would enhance the databases workability for modern day users. Reasons to retain data changes as does the method of storing the data.

C. MDBMS IMPROVEMENT POSSIBILITIES

The current MDBMS can only process up to two types of media data: sound and image. Further work in processing more media types such as graphics, video or signals would broaden the range of the media portion of the project To process multiple images and/or sound data types for one tuple. An example is having a photo of the ship and the weapons that are on the ship all in the ship relational tuple. Or, having the captain and the executive officer recorded voices accompanying the ship relational tuple. This will take some restructuring of the modification procedures and functions. Maintaining the accountability of the pointers and cursors will be the largest obstacle to overcome for these suggested improvements. More media data types integrated into the MDBMS will prove different styles of creating relations may work better than the ideas presented here on how to modify data existing in a database management system.

A user's manual to accompany the prototype would be ideal for this demonstration model. It is not easy to work with in the line by line user entry configuration without some basic knowledge of the relations and the terminology of the system. A user's manual and better exception handling are ways to improve the prototype. Increasing the error control and checking mechanisms of the prototype would help to eliminate some errors that occur at the time the user interacts with the system. At the present, not all errors are handled in the best manner. Having more checks and clear, more concise system responses for the user to interpret would only enhance the system usefulness.

As more work is done with the project, an upgrade to a current INGRES version would eliminate some of the inherent problems we encountered in working with the prototype. As mentioned in the body of the thesis, much software engineering is necessary to upgrade the code to a newer INGRES version. Work is planned to upgrade the natural language parser with a more reliable content search success. A larger dictionary caption base and possibly using super groups for the caption searches and having a text type are other prototype enhancing ideas.

APPENDIX A: SOURCE CODE FOR DELETION

```
/************************Retrieve.c*******************************
* Title          : Retrieve.c                              *
* Author         : Aygun/ Stewart                             *
* Date           : August 1991                               *
* History        : Improvements on Retrieval operations to include complex   *
*                : query processing. Also contains the procedures to perform  *
*                : the deletion operations. An if clause provides the ability *
*                : to switch options for retrieval or deletion of data      *
* Description    : This module implements the retrieval process in the       *
*                : Multimedia Database System.                       *

*************************************************************************
* Export Interface :                                     *
*     retrieve(RTRVE_MODE):                                  *
*           incorporates the retrieval process. The user is asked to   *
*           enter the name of table(s) and attribute(s) he wants to    *
*           retrieve. If he does not know the names of the tables or   *
*           attributes, he can type "?" to list all the tables and     *
*           attributes in the catalog.                         *
*     retrieve(DEL_MODE):                                     *
*           incorporates the deletion process. The user is asked to    *
*           enter the name of the relation table one at a time of what *
*           attributes in the catalog he wants to delete.            *
***********************************************************************
* Import Interface :                                     *
*     print_all_table() : Prints out the table catalog information on screen *
*                 from InsertModule.c                          *
*                                                        *
*     check_table_name(): Checks the table_name if it is duplicate        *
*     get_media_name()  : Get media table name by appending table_key at the *
*                 end of att_name.                           *
*                 from CreateModule.c                          *
*                                                        *
*     yes_no_answer()  : Gets yes or no answer from the user.            *
*     clr_scr()        : Clears the screen.                       *
*                 from UserInterface.c                         *
*                                                        *
*     play_sound(filename):Sends command from SUN to PC to play the SOUND    *
*                 media file.                              *
*                 from SoundModule.c                           *
***********************************************************************/
```

```
#include <stdio.h>
#include <string.h>
#include <pixrect/pixrect_hs.h>
#include <sys/wait.h>
#include <suntool/sunview.h>
#include <suntool/canvas.h>

#include "defines.h"
#include "errors.h"
#include "struct.h"
#include "GlobalVariables.h"

#include <rpc/rpc.h>
#include "plcall.h"
#include "defines.h"
#include "errors.h"


char c;
char temp_media_name[3];
char join_condition[100];

int look_more=0;            /* use for loop the cursor */

struct select_att satt[10];
struct select_tab stab[10];
struct group group_count[10];
int o,p,k,numcon,numgroup,icond;
STR_name tab[10];
char *all_condition;
char condition[100];

/* Selection attribute */
/* Condition attribute */
STR_name att[10];

/* Each group of attribute */
int att_group[10];

/* Condition type of each attribute 0 for formatted 1 for image 2 for sound*/
int contype[10];

/* Media attribute for description */
STR_name media_att[10];
int number_media;
```

```
/* Condition for each attribute */
char con[10][100];

/* Attribute type for each select */
STR_name atttype[10];
int cond,gcond,i_cond[10],m=0,x=0,y=0,n=0,o=0;
char buff[100],a,yes_no_answer();
char temp_table[20];
char temp_table1[20];
char temp_table2[20];
char temp_table3[5]={'h','u','s','i','4'};
char temp_table4[5]={'h','u','s','i','5'};
char temp_table8[5]={'h','u','s','o','1'};
char temp_table9[5]={'h','u','s','o','2'};
char temp_table10[5]={'h','u','s','o','3'};
char temp_table11[5]={'h','u','s','o','4'};

char group1[3]={'g','r','1'};
char group2[3]={'g','r','2'};

char condition_for_nested[100];
char attribute_for_nested[20];
char join_for_nested[99];
int more_selections;
int more_levels;
int aggregate_found;
char t1[5]={'t','_','a','_','1'};
char t2[5]={'t','_','a','_','2'};
char t3[5]={'t','_','a','_','3'};
char t4[5]={'t','_','a','_','4'};
char wrong_descrp = TRUE;
int act_media_count;
int act_media_list[10];
int media_counter=0;
int formatted_flag;
int image_flag;
int sound_flag;

/**************************************************************************/
/* Procedure initialize the array to empty                          */
/* Initialize all parameters used in the retrieve to null           */
/**************************************************************************/
void init()
{
int i,j;
```

```
icond=0;
gcond=0;
numgroup=0;
numcon=0;
for (i=0;i<10;i++) {
for (j=0;j < 13;j++) {
satt[i].t_name[j] =0;
satt[i].a_name[j] =0;
stab[i].t_name[j] =0;
att[i][j]=0;
tab[i][j]=0;
}
for (j=0;j<100;j++) {
con[i][j]='0';
}
}
}
/*************************************************************************/
/* This procedure get the table name, attribute name of that table      */
/* and then return the attribute type to the user       */
/*************************************************************************/
getatttype(tab_name,att_name,att_type)
STR_name tab_name;
STR_name att_name;
STR_name att_type;
{
  int i,j,k,found,count;
  found = 0;
  for (i=0;i < table_count;i++) {
if (strcmp(table_array[i].table_name,tab_name)==0) {
  j = table_array[i].att_entry;
  count = table_array[i].att_count;
  i = 1000;
}
    }
    for ( k=0;k < count;k++) {
if (strcmp(att_array[j].att_name, att_name)==0) {
  strcpy(att_type,att_array[j].data_type);
/* For test only */
  printf("\n%s",att_array[j].att_name);
  printf("\t%s\n",att_type);
  found = 1;
  k = 1000;
}
j = att_array[j].next_index;
    }
```

```
}
/**************************************************************************/
/* procedure to process the sound condition   */
/* put the result in the media tale [number condition] for process later  */
/**************************************************************************/
void process_icon3(query_phrase,number)
char query_phrase[DESCRLEN+1];
int number;
{
 int id;
 char answer, repeat, yes_no_answer (),con_number,medianum;
 int i, query_err, query_len, in_len, f_flag,found;
 struct pixrect *pr;
 colormap_t cm;
 char descr[DESCRLEN+1];
 int show_pid, wait_pid;
 union wait status;
 int imageno;
 printf ("\nEntering RETRIEVE ...\n");
 cm.type = RMT_NONE;
 cm.length = 0;
 cm.map[0] = NULL;
 cm.map[1] = NULL;
 cm.map[2] = NULL;
 /* this is absolutely necessary!!!! Otherwise pr_load_colormap might
    not allocate storage for the colormap, if the garbage found in
    the cm structure seems to make sense. The result, of course, is
    segmentation fault. This bug was very hard to find. */
  {
/* # line 193 "p2.sc" *//* create table */
  {
   IIsqInit((char *)0);
   IIwritedb("create ");
   temp_media_name[0]='p';
   medianum=number+48;
   temp_media_name[1]=medianum;
   temp_media_name[2]=0;
   printf("\n%s",temp_media_name);
   IIwritedb(temp_media_name);
   IIwritedb("(");
   IIwritedb("s_id=i4)");
   IIsqSync(0,(char *)0);
  }
/* # line 194 "p2.sc" *//* host code */
      printf("The query description now is:\n>>%s<<\n\n",query_phrase);
printf ("Searchqing .....\n");
```

```
   /* exec sql declare c1 cursor for
         select i_id, PIXRECT (i_image), COLORMAP (i_image),
        DESCRIPTION (i_image)
         from  emp_img1
         where SHOWS (i_image, query_phrase);
The statement is deleted by the preprocessor.
     However, the output functions and the selection conditions
associated with the cursor c1 will be used later.
The following declarations are generated: */
     {
    int ISerrorc1;
     char ISerrmcc1[ERRLEN+1];
  char *ISfnc1[FILENAMELEN + 1];
  char *ISdescrc1[DESCRLEN + 1];
     sqlca.sqlcode = 0;
     ISerrmcc1[0] = '\0';
   /* exec sql open c1; */
   /* exec sql whenever not found go to closec1; */
   /* translated by preprocessor into: */
     if ( ISerrorc1 = ISshows_open("image","i_image",ISfnc1,query_phrase,ISerrmcc1) )
     {
       sqlca.sqlcode = ISerrorc1;
  if ( sqlca.sqlcode == QUERY_WORD_ERR ||
sqlca.sqlcode == QUERY_STRUCTURE_ERR )
         strcpy(sqlca.sqlerrm.sqlerrmc,ISerrmcc1);
     }
   /* end of preprocessor output for open c1 */
     if ( !sqlca.sqlcode )
     {
       f_flag = 0;
       for (;;)
       {
      /* exec sql fetch c1
   into :imageno, :pr, :cm, :descr;
        This is translated by the preprocessor into: */
        if ( ISerrorc1 = ISshows_fetch("image","i_image",ISfnc1,query_phrase,ISerrmcc1) )
          sqlca.sqlcode = ISerrorc1;
        /*  printf("main.sc(ISfnc1):  %s\n", ISfnc1); */
        if ( sqlca.sqlcode == NOT_FOUND )
          goto closec1;
        f_flag = 1;
        if ( !sqlca.sqlcode )
        {
/* # line 653 "p1.sc" *//* select */
  strcpy (table_array[table_index].table_name, tab[number]);
  found = check_table_name();
```

```
    table_cursor = table_entry;
    strcpy(media_name,att[number]);
    get_media_name();
    printf("%s",media_name);
    {
     IIsqInit(&sqlca);
     IIwritedb("retrieve(imageno=");
     IIwritedb(media_name);
     IIwritedb(".s_id,ISdescrc1=");
     IIwritedb(media_name);
     IIwritedb(".descrp)w");
     IIwritedb("here ");
     IIwritedb(media_name);
     IIwritedb(".f_id=");
     IIsetdom(1,32,0,ISfnc1);
     IIwritedb(" ");
     IIsqRinit(&sqlca);
     if (IIerrtest() == 0) {
      if (IInextget() != 0) {
       IIretdom(1,30,4,&imageno);
       IIretdom(1,32,0,ISdescrc1);
      } /* IInextget */
      IIsqFlush(&sqlca);
     } /* IIerrtest */
    }
/* # line 657 "p1.sc" *//* host code */
            if (!sqlca.sqlcode)
            {
             ISerrorc1 = ISdescription (ISfnc1, ISdescrc1, descr);
       sqlca.sqlcode = ISerrorc1;
       }
            else
              sqlca.sqlcode = PROGRAM_ERR;
          }
  /* end of preprocessor output for fetch c1 */
     if (sqlca.sqlcode)
       goto closec1;
     id = imageno;
/* # line 270 "p2.sc" *//* insert */
    {
     IIsqInit((char *)0);
     IIwritedb("append to ");
     IIwritedb(temp_media_name);
     IIwritedb("(s_id=");
     IIsetdom(1,30,4,&id);
     IIwritedb(" )");
```

```
  IIsqSync(3,(char *)0);
 }
/* # line 272 "p2.sc" *//* host code */
        } /* end for loop of cursor c1 */
       closec1:
 /* exec sql close c1; */
 /* translated by the preprocessor into: */
        sqlca.sqlcode = ISshows_close("image","i_image",ISfnc1,query_phrase,ISerrmcc1);
/* # line 693 "p1.sc" *//* host code */
}  /* end of successful open c1; correct query description  */
 }  /* end of preprocessor declaration block */
      if ( sqlca.sqlcode == QUERY_WORD_ERR )
      {
        printf("The system cannot understand the word >>%s<<\n",sqlca.sqlerrm.sqlerrmc);
        query_err = 1;
      }
      if ( sqlca.sqlcode == QUERY_STRUCTURE_ERR )
      {
        printf("The system cannot interpret the phrase\n>>\n%s<<\n",sqlca.sqlerrm.sqlerrmc);
        query_err = 1;
      }
      if ( query_err )
      {
      }
    }
    if ( !f_flag )
      printf("There are no media matching that query description.\n");
    if ( sqlca.sqlcode )
      printf("An error has occured while accessing the database\n\
sql error code: %d\n", sqlca.sqlcode);
   clr_scr();
}  /* end of retrieve_photo () */
/***************************************************************************/
/* procedure to process the image condition  */
/* put the result in the media tale [number condition] for process later */
/***************************************************************************/
void process_icon2(query_phrase,number)
char query_phrase[DESCRLEN+1];
int number;
{
 int id;
 char answer, repeat, yes_no_answer (),con_number,medianum;
 int i, query_err, query_len, in_len, f_flag,found;
 struct pixrect *pr;
 colormap_t cm;
 char descr[DESCRLEN+1];
```

```
 int show_pid, wait_pid;
 union wait status;
 int imageno;
 printf ("\nEntering RETRIEVE ...\n");
 cm.type = RMT_NONE;
 cm.length = 0;
 cm.map[0] = NULL;
 cm.map[1] = NULL;
 cm.map[2] = NULL;
 /* this is absolutely necessary!!!! Otherwise pr_load_colormap might
    not allocate storage for the colormap, if the garbage found in
    the cm structure seems to make sense. The result, of course, is
    segmentation fault. This bug was very hard to find. */
  {
/* # line 193 "p2.sc" *//* create table */
  {
   IIsqInit((char *)0);
   IIwritedb("create ");
   temp_media_name[0]='p';
   medianum=number+48;
   temp_media_name[1]=medianum;
   temp_media_name[2]=0;
   printf("\n%s",temp_media_name);
   IIwritedb(temp_media_name);
   IIwritedb("(");
   IIwritedb("i_id=i4)");
   IIsqSync(0,(char *)0);
  }
/* # line 194 "p2.sc" *//* host code */
      printf("The query description now is:\n>>%s<<\n\n",query_phrase);
printf ("Searching .....\n");
    /* exec sql declare c1 cursor for
          select i_id, PIXRECT (i_image), COLORMAP (i_image),
         DESCRIPTION (i_image)
          from  emp_img1
          where SHOWS (i_image, query_phrase);
The statement is deleted by the preprocessor.
      However, the output functions and the selection conditions
associated with the cursor c1 will be used later.
The following declarations are generated: */
      {
     int ISerrorc1;
      char ISerrmcc1[ERRLEN+1];
  char ISfnc1[FILENAMELEN + 1];
  char ISdescrc1[DESCRLEN + 1];
      sqlca.sqlcode = 0;
```

```
      ISerrmcc1[0] = '\0';
    /* exec sql open c1; */
    /* exec sql whenever not found go to closec1; */
    /* translated by preprocessor into: */
      if ( ISerrorc1 = ISshows_open("image","i_image",ISfnc1,query_phrase,ISerrmcc1) )
      {
        sqlca.sqlcode = ISerrorc1;
   if ( sqlca.sqlcode == QUERY_WORD_ERR ||
sqlca.sqlcode == QUERY_STRUCTURE_ERR )
          strcpy(sqlca.sqlerrm.sqlerrmc,ISerrmcc1);
      }
    /* end of preprocessor output for open c1 */
      if ( !sqlca.sqlcode )
      {
        f_flag = 0;
        for (;;)
        {
       /* exec sql fetch c1
    into :imageno, :pr, :cm, :descr;
         This is translated by the preprocessor into: */
          if ( ISerrorc1 = ISshows_fetch("image","i_image",ISfnc1,query_phrase,ISerrmcc1) )
            sqlca.sqlcode = ISerrorc1;
          /* printf("main.sc(ISfnc1): %s\n", ISfnc1); */
          if ( sqlca.sqlcode == NOT_FOUND )
            {
             printf("main.sc: ISshows_fetch liefert NOT_FOUND");
             goto closec1;
            }
          f_flag = 1;
          if ( !sqlca.sqlcode )
          {
/* # line 653 "p1.sc" *//* select */
  strcpy (table_array[table_index].table_name, tab[number]);
  found = check_table_name();
  table_cursor = table_entry;
  strcpy(media_name,att[number]);
  get_media_name();
  printf("%s",media_name);
  {
    IIsqInit(&sqlca);
    IIwritedb("retrieve(imageno=");
    IIwritedb(media_name);
    IIwritedb(".i_id,ISdescrc1=");
    IIwritedb(media_name);
    IIwritedb(".descrp)w");
    IIwritedb("here ");
```

```
   IIwritedb(media_name);
   IIwritedb(".f_id=");
   IIsetdom(1,32,0,ISfnc1);
   IIwritedb(" ");
   IIsqRinit(&sqlca);
   if (IIerrtest() == 0) {
    if (IInextget() != 0) {
     IIretdom(1,30,4,&imageno);
     IIretdom(1,32,0,ISdescrc1);
    } /* IInextget */
    IIsqFlush(&sqlca);
   } /* IIerrtest */
  }
/* # line 657 "p1.sc" *//* host code */
           if (!sqlca.sqlcode)
           {
        if (!(ISerrorc1 = ISpixrect (ISfnc1, ISdescrc1, &pr)))
          if (!(ISerrorc1 = IScolormap (ISfnc1, ISdescrc1, &cm)))
            ISerrorc1 = ISdescription (ISfnc1, ISdescrc1, descr);
      sqlca.sqlcode = ISerrorc1;
      }
           else
             sqlca.sqlcode = PROGRAM_ERR;
         }
  /* end of preprocessor output for fetch c1 */
    if (sqlca.sqlcode)
      goto closec1;
    id = imageno;
/* # line 270 "p2.sc" *//* insert */
  {
   IIsqInit((char *)0);
   IIwritedb("append to ");
   IIwritedb(temp_media_name);
   IIwritedb("(i_id=");
   IIsetdom(1,30,4,&id);
   IIwritedb(" )");
   IIsqSync(3,(char *)0);
  }
/* # line 272 "p2.sc" *//* host code */
        } /* end for loop of cursor c1 */
      closec1:
  /* exec sql close c1; */
  /* translated by the preprocessor into: */
        sqlca.sqlcode = ISshows_close("image","i_image",ISfnc1,query_phrase,ISerrmcc1);
/* # line 693 "p1.sc" *//* host code */
 }   /* end of successful open c1; correct query description  */
```

```
}  /* end of preprocessor declaration block */
    if ( sqlca.sqlcode == QUERY_WORD_ERR )
    {
      printf("The system cannot understand the word >>%s<<\n",sqlca.sqlerrm.sqlerrmc);
      query_err = 1;
    }
    if ( sqlca.sqlcode == QUERY_STRUCTURE_ERR )
    {
      printf("The system cannot inte pret the phrase\n>>\n%s<<\n",sqlca.sqlerrm.sqlerrmc);
      query_err = 1;
    }
    if ( query_err )
    {
    }
  }
  if ( !f_flag )
    printf("There are no media matching that query description.\n");
  if ( sqlca.sqlcode )
    printf("An error has occured while accessing the database\n\
sql error code: %d\n", sqlca.sqlcode);
  clr_scr();
} /* end of retrieve_photo () */
/*****************************************************************/
/* This procedure search through the media relation and get the   */
/* file name that match with the result table and send to the     */
/* present photo procedure                                        */
/*****************************************************************/
display_photo (imageno,tupleno,temp_table, image_id)
int imageno;
int tupleno;
char temp_table[20];
int image_id;
{
 int desired_tupleno;
 char image_value[20];
 char answer, repeat, yes_no_answer ();
 char query_phrase[DESCRLEN+1],
    in_phrase[DESCRLEN+1];
 int i=0,j=0, k, c, pid, query_err, query_len, in_len, f_flag,look_more=0;
 struct pixrect *pr;
 colormap_t cm;
 char ISfn1[FILENAMELEN+1];
 char descr[DESCRLEN+1];
 int show_pid, wait_pid;
 int  ISerror;
 STR_path file_name;
```

```
 char ISdescr1[DESCRLEN+1];
 cm.type = RMT_NONE;
 cm.length = 0;
 cm.map[0] = NULL;
 cm.map[1] = NULL;
 cm.map[2] = NULL;
 desired_tupleno=tupleno;
 /* this is absolutely necessary!!!! Otherwise pr_load_colormap might
   not allocate storage for the colormap, if the garbage found in
   the cm structure seems to make sense. The result, of course, is
   segmentation fault. This bug was very hard to find. */
  /*  exec sql select PIXRECT (i_image), COLORMAP (i_image),
                DESCRIPTION (i_image)
            into :pr, :cm, :descr
          from image
      where i_id = :imageno;
    This Image-SQL statement is transformed into the following
    sequence of statements by the preprocessor:
  */
c=1;
inttostr(image_id, image_value);
{
if (IIcsrOpen((char *)0,"cursor_output1","db",0,media_name) != 0) {
   IIwritedb("retrieve(ISfn1=");
   IIwritedb(media_name);
   IIwritedb(".");
   IIwritedb("f_id,ISdescr1=");
   IIwritedb(media_name);
   IIwritedb(".descrp");
   IIwritedb(")where ");
   IIwritedb(media_name);
   IIwritedb(".i_id=");
   IIwritedb(image_value);
   IIcsrQuery ((char *)0);
  }


while (look_more==0) {
 if (IIcsrFetch((char *)0, "cursor_output1","db") != 0) {
   IIcsrRet(1,32,0,ISfn1);
   IIcsrRet(1,32,0,ISdescr1);
   for (i=0;i<MAX_PATH+1;i++) {
    if (ISfn1[i]==32) {
file_name[i]=0;
    }
    else {
```

```
file_name[i]=ISfn1[i];
    }
   }
   printf("\nRecord no %d filename :%s:",j+1, ISfn1);
   if ((img_file=fopen(file_name,"r"))==NULL)
    {
printf("\n%s", file_name);
printf("\nThe file cannot be opened !\n");
putchar('\007');
    }
   else {
     pr=pr_load(img_file, &cm);
     if (pr==NULL) {
printf("\nThe file does not contain proper image");
putchar('\007');
    }
     else {
  printf("\nShow image ....");
  present_photo(j+1,pr,&cm,ISdescr1);
  IIcsrClose((char *)0,"cursor_output1","db");
    }
   }
   fclose(img_file);
  }
  IIcsrEFetch((char *)0);
  j++;
  if (j==c) {
   look_more = 1;
  };
}
/*IIcsrClose((char *)0,"cursor_output1","db");*/
}
}
/*******************************************************************/
/* This procedure search through the media relation and get the    */
/* file name that match with the result table and send to the      */
/* play sound procedure                                            */
/*******************************************************************/
display_sound (soundno,tupleno,temp_table, sound_id)
int soundno;
int tupleno;
char temp_table[20];
int sound_id;
{
  char sound_value[20];
  int desired_tupleno;
```

```
char Answer,answer, repeat, yes_no_answer();
char query_phrase[DESCRLEN+1],
    in_phrase[DESCRLEN+1];
int i=0,j=0, k, c, pid, query_err, query_len, in_len, f_flag,look_more=0;
int show_pid, wait_pid;
int  ISerror;
STR_path file_name;
char ISfn1[FILENAMELEN+1];
char ISdescr1[DESCRLEN+1];
desired_tupleno=tupleno;
c=1;

inttostr(sound_id, sound_value);
if (IIcsrOpen((char *)0,"cursor_output1","db4",0,media_name) != 0) {
  IIwritedb("retrieve(ISfn1=");
  IIwritedb(media_name);
  IIwritedb(".");
  IIwritedb("f_id,ISdescr1=");
  IIwritedb(media_name);
  IIwritedb(".descrp");
  IIwritedb(")where ");
  IIwritedb(media_name);
  IIwritedb(".s_id=");
  IIwritedb(sound_value);
  IIcsrQuery ((char *)0);
 } /* IIcsropen */
 while (look_more==0) {
  if (IIcsrFetch((char *)0, "cursor_output1","db4") != 0) {
   IIcsrRet(1,32,0,ISfn1);
   IIcsrRet(1,32,0,ISdescr1);
   for (i=0;i<MAX_PATH+1;i++) {
if (ISfn1[i]==32) {
 file_name[i]=0;
}
else {
 file_name[i]=ISfn1[i];
}
   }
   printf("\nRecord no %d ",j+1);
   printf("\nPlay the sound ? (y/n) :: ");
   if (yes_no_answer()=='y'){
play_sound(file_name);
IIcsrClose((char *)0,"cursor_output1","db4");
   }
   IIcsrEFetch((char *)0);
   j++;
```

```
    if (j==c) {
look_more = 1;
    }
   } /* IICSRFECCH */
  } /* end while */

}  /* end of display_sound () */
/*******************************************************************/
/* This procedure get the query description for the media attribute*/
/* from the user phrase by phrase                        */
/*******************************************************************/
char process_icon()
{
 char answer, repeat, yes_no_answer ();
 char query_phrase[DESCRLEN+1],
     in_phrase[DESCRLEN+1];
 int i, query_err, query_len, in_len, f_flag;
 char descr[DESCRLEN+1];
 int show_pid, wait_pid;
 int imageno;
 icond = 1;
 do
 {
query_err = 0;
query_len = 0;
query_phrase[0] = '\0';
printf("\nPlease enter your query description\n\
 * noun phrases separate by commas and end with an exclamation mark\n\
 * sentence end with a period.\n\
(end whole description with an empty line):\n");
do /* until query_phrase input */
{
  i = 0;
  while ( (in_phrase[i++] = getchar()) != '\n' && i < 127 );
  if ( in_phrase[i-1] != '\n' )
  {
    in_phrase[i-1] = '\n';
    printf ("The phrase is too long, it will be shortened\n");
    while ( getchar () != '\n' );
  } /* End if */
  in_phrase[i] = '\0';
  if ( ( in_len = i ) > 1 )
  {
    if ( query_len + in_len < DESCRLEN )
    {
 strcat(query_phrase,in_phrase);
```

```
 query_len = query_len + in_len;
     } /* End if */
     else
     {
 printf("The last phrase extended beyond the maximum \
description length,\nit will be ignored\n");
 break;
     }   /* End else */
   } /* End if */
   if ( !query_len )
     printf("\nAn empty string is not allowed as a query description.\n\
Please type at least a single word:\n");
} /* End do */
while ( ( in_len > 1 ) || !query_len ); /* end query_phrase input */
printf("The query description now is:\n>>%s<<\n\n",query_phrase);
} while (query_err);
   strcpy(con[numcon],query_phrase);
   if (contype[numcon]==1) {
   process_icon2 (query_phrase,numcon);
}
   if (contype[numcon]==2) {
   process_icon3 (query_phrase,numcon);
}

}
/**********************************************************
This procedure handles if there are  more than one conditions in the query.
**********************************************************/
nested_gcondition(choice,temp_table1,temp_table2,temp_table)
char choice;
char temp_table1[20];
char temp_table2[20];
char temp_table[20];
{
 int group_number=0;
 int nested_counter=0;
 int k,l;
int endgroup,i,more,found=FALSE;
char ans,ans2;
 endgroup = 0;
 more = 0;
 numcon=0;
 numgroup=0;

 choice=utility_menu(choice,temp_table1,temp_table2,temp_table);
```

```
 if (choice=='0'){
  cond=1;
  gcond=0;
 }
while (more != 1) {
while (endgroup != 1) {
for (i=0;i < att_index;i++)
{
 if (choice=='0'){
  if (m > 1 ) {
   printf("\nEnter table name ");
   gets(tab[numcon]);
   strcpy (table_array[table_index].table_name, tab[numcon]);
  }
 }
if (m==1) {
  strcpy (tab[numcon], stab[0].t_name);
 }
if (choice=='0'){
 cond=1;
 gcond=0;
 printf("\nEnter attribute ");
 gets(att[numcon]);
 getatttype(tab[numcon], att[numcon],atttype[numcon]);
 if (strcmp(atttype[numcon],"image")==0)
  {
   contype[numcon]=1;
   process_icon();
  }
 else if (strcmp(atttype[numcon],"sound")==0)
  {
   contype[numcon]=2;
   process_icon();
  }
 else {
  printf("Enter the condition \n");
  gets(con[numcon]);
  contype[numcon]=0;
 }
}/*end if choice==0 */

 nested_counter=nested_counter+1;
 if ((nested_counter%2)==1){
  if (choice=='0'){
   cond=1;
   gcond=0;
```

```
    ql_retrieve(temp_table8);
/*    ql_printdata(temp_table8);*/
    cond=0;
    numcon=0;
    numgroup=0;
    init_buffer(tab,10);
    init_buffer(att,10);
    for (k=0; k<10; k++){
for (l=0; l<100; l++){
 con[k][l]='0';
}
    }
   }
   if (choice=='1'){
    temp1_exists_temp2(temp_table1, temp_table2, temp_table8);
    ql_printdata(temp_table8);
    init_buffer(join_for_nested,99);
   }
   if (choice=='2'){
    temp1_not_exists_temp2(temp_table1, temp_table2, temp_table8);
    ql_printdata(temp_table8);
    init_buffer(join_for_nested,99);
   }
   if (choice=='3'){
    temp1_in_temp2(temp_table1, temp_table2, temp_table8);
    ql_printdata(temp_table8);
    init_buffer(condition_for_nested,100);
    init_buffer(attribute_for_nested,20);
   }
   if (choice=='4'){
    temp1_not_in_temp2(temp_table1, temp_table2, temp_table8);
    ql_printdata(temp_table8);
    init_buffer(condition_for_nested,100);
    init_buffer(attribute_for_nested,20);

   }
  }/* end if nested_counter%2==1 */


  if ((nested_counter%2)==0){
   if (choice=='0'){
    cond=1;
    gcond=0;
    ql_retrieve(temp_table9);
/*    ql_printdata(temp_table9);*/
    cond=0;
```

```
    numcon=0;
    numgroup=0;
    init_buffer(tab,10);
    init_buffer(att,10);
    for (k=0; k<10; k++){
for (l=0; l<100; l++){
 con[k][l]='0';
}
    }
   }
   if (choice=='1'){
    temp1_exists_temp2(temp_table1, temp_table2, temp_table9);
    ql_printdata(temp_table9);
    init_buffer(join_for_nested,99);
   }
   if (choice=='2'){
    temp1_not_exists_temp2(temp_table1, temp_table2, temp_table9);
    ql_printdata(temp_table9);
    init_buffer(join_for_nested,99);
   }
   if (choice=='3'){
    temp1_in_temp2(temp_table1, temp_table2, temp_table9);
    ql_printdata(temp_table9);
    init_buffer(condition_for_nested,100);
    init_buffer(attribute_for_nested,20);
   }
   if (choice=='4'){
    temp1_not_in_temp2(temp_table1, temp_table2, temp_table9);
    ql_printdata(temp_table9);
    init_buffer(condition_for_nested,100);
    init_buffer(attribute_for_nested,20);
   }
  }/* end if nested_counter%2==0 */


if (nested_counter==2){
/*    printf("\nBelow is the result of the first %d  conditions in group %d :", nested_counter,
group_number+1);*/
/* printf("\nBefore intersection...nested_counter->%d",nested_counter);*/
 intersect_tables(temp_table8,temp_table9,temp_table10);
/* ql_printdata(temp_table10);*/
 drop_table(temp_table8);
 drop_table(temp_table9);
}
```

```
if (nested_counter>2){
 if ((nested_counter%2)==1){
/*    printf("\nBelow is the result of the first %d  conditions in group %d :", nested_counter,
group_number+1);*/
/*   printf("\nBefore intersection...nested_counter->%d",nested_counter);*/
   intersect_tables(temp_table10,temp_table8,temp_table11);
/*   ql_printdata(temp_table11);*/
   drop_table(temp_table8);
   drop_table(temp_table10);
 }

 if ((nested_counter%2)==0){
/*    printf("\nBelow is the result of the first %d  conditions in group %d :", nested_counter,
group_number+1);*/
/*   printf("\nBefore intersection...nested_counter->%d",nested_counter);*/
   intersect_tables(temp_table11,temp_table9,temp_table10);
/*   ql_printdata(temp_table10);*/
   drop_table(temp_table11);
   drop_table(temp_table9);
 }
}/* end if nested_counter>2 */

/*}*/ /* end if '1'<=choice=<'4' -----deneme-----*/

printf("\nEnd group ?");
ans=yes_no_answer();
if ((ans==121)||(ans==89)) {
 group_number=group_number+1;
 if (group_number==1){
   if (nested_counter==1){
     union_tables_for_nested(temp_table8, group1);
/*    printf("\nBelow is the result of group %d :",group_number);
     ql_printdata(group1);*/
     drop_table(temp_table8);
   }
   if (nested_counter>1){
     if ((nested_counter%2)==0){
union_tables_for_nested(temp_table10, group1);
/*    printf("\nBelow is the result of group %d :",group_number);
ql_printdata(group1);*/
drop_table(temp_table10);
     }
     if ((nested_counter%2)==1){
union_tables_for_nested(temp_table11, group1);
/*    printf("\nBelow is the result of group %d :",group_number);
ql_printdata(group1);*/
```

```
drop_table(temp_table11);
    }
   }/* end if nested_counter > 1 */
  }/*end if group_number==1 */

  if (group_number==2){
   if (nested_counter==1){
    union_tables_for_nested(temp_table8, group2);
/*    printf("\nBelow is the result of group %d :",group_number);
    ql_printdata(group2);*/
    drop_table(temp_table8);
   }
   if (nested_counter>1){
    if ((nested_counter%2)==0){
union_tables_for_nested(temp_table10, group2);
/*    printf("\nBelow is the result of group %d :",group_number);
ql_printdata(group2);*/
drop_table(temp_table10);
    }
    if ((nested_counter%2)==1){
union_tables_for_nested(temp_table11, group2);
/*    printf("\nBelow is the result of group %d :",group_number);
ql_printdata(group2);*/
drop_table(temp_table11);
    }
   }/* end if nested_counter>1 */
   union_tables(group1, group2);
/*   printf("\nBelow is the result of the first %d groups ",group_number);
   ql_printdata(group2);*/
   drop_table(group1);
  }/*end if group_number==2 */

  if (group_number>2){
   if (nested_counter==1){
    union_tables_for_nested(temp_table8, group1);
/*    printf("\nBelow is the result of group %d :",group_number);
    ql_printdata(group1);*/
    drop_table(temp_table8);
   }
   if (nested_counter>1){
    if ((nested_counter%2)==0){
union_tables_for_nested(temp_table10, group1);
/*printf("\nBelow is the result of group %d :",group_number);
ql_printdata(group1);*/
drop_table(temp_table10);
    }
```

```
    if ((nested_counter%2)==1){
union_tables_for_nested(temp_table11, group1);
/*printf("\nBelow is the result of group %d :",group_number);
ql_printdata(group1);*/
drop_table(temp_table11);
     }
   }/* end if nested_counter>1 */
   union_tables(group1,group2);
/*   printf("\nBelow is the result of the first %d groups ",group_number);
   ql_printdata(group2);*/
   drop_table(group1);
  }/* end if group_number > 2 */
  nested_counter=0;
  endgroup=1;
 /* printf("\nGroup      %d",numgroup);
  printf("\nCondition  %d",numcon);*/
  i=600;
 }/* end if ans= YES to end group ? */
 if ((ans==110)||(ans==78)) {
  choice=utility_menu(choice,temp_table1,temp_table2,temp_table);
 }
     } /* End for */
     } /* END WHILE */
printf("\nEnd condition ?");
ans=yes_no_answer();
if ((ans==121)||(ans==89))
  {
   if (group_number==1){
     union_tables_for_nested(group1, temp_table);
     drop_table(group1);
     printf("\nBelow is the final result :");
     ql_printdata(temp_table);
   }
   if (group_number>1){
     union_tables_for_nested(group2, temp_table);
     drop_table(group2);
     printf("\nBelow is the final result :");
     ql_printdata(temp_table);
   }
/*   if (choice=='0')
     group_count[numgroup].endgroup = numcon-1;*/
   endgroup=1;
   more = 1;
   i=0;
  }/* if ans=YES to end condition? */
 else {
```

```
 more=0;
 endgroup=0;
 i=0;
 nested_counter=0;
 choice=utility_menu(choice,temp_table1,temp_table2,temp_table);
/*   group_count[numgroup].endgroup = numcon-1;
  numgroup=numgroup+1;
  group_count[numgroup].begingroup=numcon;*/

}/*end else*/

    } /* End more */
 group_number=0;
}
/*********************************************************************
This function handles if there is only one condition in the query.
*********************************************************************/
nested_processcondition(choice,temp_table1,temp_table2,temp_table)
char choice;
char temp_table1[20];
char temp_table2[20];
char temp_table[20];
{
char   ans2,a;
int i,j;
gcond=0;
printf("\nGroup condition ? (y/n)  ");
ans2=yes_no_answer();
if ((ans2==121)||(ans2==89))
{
 nested_gcondition(choice,temp_table1,temp_table2,temp_table);
}
else
{

gcond=0;
choice=utility_menu(choice,temp_table1,temp_table2,temp_table);
if (choice == '0'){
 cond=1;
 if (m > 1 ) {
  printf("\nEnter table name ");
  gets(tab[0]);
 }
 if (m==1) {
  strcpy (tab[0], stab[0].t_name);
 }
```

```
 printf("\nEnter attribute name ");
 gets(att[0]);
 printf("\n%s %s %s", tab[0], att[0], atttype[0]);
 getatttype(tab[0],att[0],atttype[0]);
 if (strcmp(atttype[0],"image")==0)
  {
   contype[0]=1;
   process_icon();
  }
 else if (strcmp(atttype[0],"sound")==0)
  {
   contype[0]=2;
   process_icon();
  }
 else {
  printf("Enter the condition \n");
  gets(con[0]);
  contype[0]=0;
 }
}
else
 cond=0;

if (choice=='0')
 ql_retrieve(temp_table);
if (choice=='1')
 temp1_exists_temp2(temp_table1, temp_table2, temp_table);
if (choice=='2')
 temp1_not_exists_temp2(temp_table1, temp_table2, temp_table);
if (choice=='3')
 temp1_in_temp2(temp_table1, temp_table2, temp_table);
if (choice=='4')
 temp1_not_in_temp2(temp_table1, temp_table2, temp_table);
ql_printdata(temp_table);
    }
 }

/*********************************************************************/
/* This procedure print the attribute name of the table assign to    */
/*********************************************************************/
void p_att(tab_name)
STR_name tab_name;
{
int i,j;
        for (i=0;i<= table_count;i++) {
   if (strcmp(table_array[i].table_name,tab_name)==0) {
```

```
x = i;
y = table_array[i].att_entry;
printf("\nTable Name: %s\n",table_array[i].table_name); /* print table name */
          printf("\n**Attribute****Data Type**");
while (y != -1) {
                printf("\n%13s    %s",att_array[y].att_name,att_array[y].data_type);
 y = att_array[y].next_index;
} /* End while y!=-1 */
if (y==-1) {
                    printf("\n");
i=500;
} /* Exit loop */
}  /* End if */
} /* End for */
}
/**************************************************/
/* Generate the result table for retrieval process */
/* This procedure process the query and condition  */
/* By using the select_array and condition_array   */
/* also group_array   */
/**************************************************/
ql_retrieve(temp_table)
char temp_table[20];
{
 int d,e,r;
  int i,j,k,l;
  char grnum,medianum,operator[4];
  i=0; /* set up index to 0 */
/* Below is the embeded C code for the SQL C for INGRES */
/* This is equivalent to the SQL query */
/*  exec sql select (var1, var2, ...)
from (table1, table2,...)
where (condition1 and/or condition2 and/or ...);
*/
  k=0;
  i=0;
  j=0;
  l=0;
 r=0;

  IIsqInit((char *)0);
  IIwritedb("retrieve into ");
  IIwritedb(temp_table);
  IIwritedb("(");
  for (i=0;i<n-1;i++) {
  IIwritedb(satt[i].t_name);
```

```
    IIwritedb(".");
    IIwritedb(satt[i].a_name);
    IIwritedb(",");
  } /* end for */
    IIwritedb(satt[i].t_name);
    IIwritedb(".");
    IIwritedb(satt[i].a_name);
    IIwritedb(")");
    if (cond==0) {
  if (m>1) {
  IIwritedb("where(");
  IIwritedb(join_condition);
  IIwritedb(")");
  }
    }
   if (cond==1) {
    IIwritedb("where(");
  if (m>1) {
  IIwritedb("(");
  IIwritedb(join_condition);
  IIwritedb(")");
  IIwritedb(" and ");
  }
  if (gcond==0) {
   if (contype[0]==0) {
  IIwritedb(tab[0]);
  IIwritedb(".");
  IIwritedb(att[0]);
  IIwritedb(con[0]);
  } /* end if */
  if (contype[0]==1) {
  IIwritedb(tab[0]);
  IIwritedb(".");
  IIwritedb(att[0]);
  IIwritedb("=");
    temp_media_name[0]='p';
    medianum=0+48;
    temp_media_name[1]=medianum;
    temp_media_name[2]=0;
    IIwritedb(temp_media_name);
  IIwritedb(".");
  IIwritedb("i_id");
  }
   if (contype[0]==2) {
  IIwritedb(tab[0]);
  IIwritedb(".");
```

```
IIwritedb(att[0]);
IIwritedb("=");
   temp_media_name[0]='p';
   medianum=0+48;
   temp_media_name[1]=medianum;
   temp_media_name[2]=0;
   IIwritedb(temp_media_name);
IIwritedb(".");
IIwritedb("s_id");
}
     } /* end if no group */
IIwritedb(")");
 } /* end if con=1 */
}
/*****************************************************************
This function takes two temp tables and unions them and returns
the result to the calling function
*****************************************************************/
union_tables(temp_table1, temp_table)
char temp_table1[20];
char temp_table[20];
{

 int c=0,j=0,k=0,l=0,temp, count;
 /*char*/ STR_name char_value[21];
 char file_name[20],a;
 int  integer_value,media_value,found,media1_value;
 float real_value;
  int i=0,select=0;
int g=0;
/*printf("\nNow we are in union_tables");*/
/* # line 3169 "db.sc" *//* select */
 {
   IIsqInit((char *)0);
   IIwritedb("retrieve(c=(count(");
   IIwritedb(temp_table1);
   IIwritedb(".");
   IIwritedb(satt[0].a_name);
   IIwritedb(")))");
   IIsqRinit((char *)0);
   if (IIerrtest() == 0) {
    if (IInextget() != 0) {
     IIretdom(1,30,4,&c);
    } /* IInextget */
    IIsqFlush((char *)0);
   } /* IIerrtest */
```

```
 }
 l=0;
 /*printf("\nThere are %d records in temp_table %s",c, temp_table1);*/

 /* # line 3171 "db.sc" *//* host code */
    if (IIcsrOpen((char *)0,"cursor_output","db1",0,temp_table1) != 0) {
      IIwritedb("retrieve(");
      for (select=0;select<n-1;select++) {
      IIwritedb(satt[select].a_name);
      IIwritedb("=");
      IIwritedb(temp_table1);
      IIwritedb(".");
      IIwritedb(satt[select].a_name);
      IIwritedb(",");
 }
      IIwritedb(satt[select].a_name);
      IIwritedb("=");
      IIwritedb(temp_table1);
      IIwritedb(".");
      IIwritedb(satt[select].a_name);
      IIwritedb(")");
      IIcsrQuery((char *)0);
    } /* IIcsrOpen */

 /* # line 3169 "db.sc" *//* select */
   {
    IIsqInit((char *)0);
    IIwritedb("retrieve(g=(count(");
    IIwritedb(temp_table);
    IIwritedb(".");
    IIwritedb(satt[0].a_name);
    IIwritedb(")))");
    IIsqRinit((char *)0);
    if (IIerrtest() == 0) {
      if (IInextget() != 0) {
        IIretdom(1,30,4,&g);
      }
      IIsqFlush((char *)0);
    }
   }

 /*printf("\nThere are %d records in temp_table %s",g, temp_table);*/

 /* # line 3171 "db.sc" *//* host code */
    if (IIcsrOpen((char *)0,"cursor_output","db2",0,temp_table) != 0) {
      IIwritedb("retrieve(");
```

```
    for (select=0;select<n-1;select++) {
    IIwritedb(satt[select].a_name);
    IIwritedb("=");
    IIwritedb(temp_table);
    IIwritedb(".");
    IIwritedb(satt[select].a_name);
    IIwritedb(",");
}
    IIwritedb(satt[select].a_name);
    IIwritedb("=");
    IIwritedb(temp_table);
    IIwritedb(".");
    IIwritedb(satt[select].a_name);
    IIwritedb(")");
    IIcsrQuery((char *)0);
   }

/*printf("\n");*/
look_more=0;
l=0;
if (c==0) {
look_more=1;
}

/* Fetch the cursor to the result relation which is the intermediate table
  hold the result from the query, then print out the tuple one at a time
  until no more record to print to the user */

/* # line 7 "insert.sc" *//* insert */
 {
  while (look_more == 0) {
   if (IIcsrFetch((char *)0,"cursor_output","db1") != 0) {
IIsqInit((char *)0);
IIwritedb("append to ");
IIwritedb(temp_table);
IIwritedb("(");
/*printf("\nrecord id %d \t",l+1);*/
for (i=0;i<n-1;i++) {
 IIwritedb(satt[i].a_name);
 IIwritedb("=");
 if (strcmp(satt[i].data_type,"c20")==0) {
  IIcsrRet(1,32,0, char_value);
/*   printf("%s : %s",satt[i].a_name,char_value);*/
  IIsetdom(1,32,0, char_value);
 }
 if (strcmp(satt[i].data_type,"integer")==0) {
```

```
    IIcsrRet(1,30,4,&integer_value);
  /*   printf("%s :  %d  ",satt[i].a_name,integer_value);*/
    IIsetdom(1,30,4,&integer_value);
   }
   if (strcmp(satt[i].data_type,"float")==0) {
    IIcsrRet(1,31,4,&real_value);
  /*   printf("%s :  %8.2f  ",satt[i].a_name,real_value);*/
    IIsetdom(1,31,4,&real_value);
   }
   if (strcmp(satt[i].data_type,"image")==0) {
    IIcsrRet(1,30,4,&media_value);
  /*   printf("%s id is %d  ",satt[i].a_name,media_value);*/
    IIsetdom(1,30,4,&media_value);
   }
   if (strcmp(satt[i].data_type,"sound")==0) {
    IIcsrRet(1,30,4,&media1_value);
  /*   printf("%s %d",satt[i].a_name,media1_value);*/
    IIsetdom(1,30,4,&media1_value);
   }
   IIwritedb(",");
  }
  IIwritedb(satt[i].a_name);
  IIwritedb("=");
  if (strcmp(satt[i].data_type,"c20")==0) {
   IIcsrRet(1,32,0, char_value);
  /* printf("%s :  %s",satt[i].a_name,char_value);*/
   IIsetdom(1,32,0, char_value);
  }
  if (strcmp(satt[i].data_type,"integer")==0) {
   IIcsrRet(1,30,4,&integer_value);
  /* printf("%s :  %d  ",satt[i].a_name,integer_value);*/
   IIsetdom(1,30,4,&integer_value);
  }
  if (strcmp(satt[i].data_type,"float")==0) {
   IIcsrRet(1,31,4,&real_value);
  /* printf("%s :  %8.2f  ",satt[i].a_name,real_value);*/
   IIsetdom(1,31,4,&real_value);
  }
  if (strcmp(satt[i].data_type,"image")==0) {
   IIcsrRet(1,30,4,&media_value);
  /* printf("%s id is %d  ",satt[i].a_name,media_value);*/
   IIsetdom(1,30,4,&media_value);
  }
  if (strcmp(satt[i].data_type,"sound")==0) {
   IIcsrRet(1,30,4,&media1_value);
  /* printf("%s %d",satt[i].a_name,media1_value);*/
```

```
 IIsetdom(1,30,4,&media1_value);
}

/*printf("\n");*/
IIcsrEFetch((char *)0);  /* fetch the next record to the cursor */
l++; /* increment l as the counter */
if (l==c) {   /* check if no more data to print */
 look_more =1;  /* exit of the loop */
}
IIwritedb(" )");
IIsqSync(3,(char *)0);
     } /* IIcsrFetch */
   } /* end while */
  }
 IIcsrClose((char *)0,"cursor_output","db1"); /* close the cursor */
 IIcsrClose((char *)0,"cursor_output","db2"); /* close the cursor */
 return(temp_table);
}
/*****************************************************************
This function takes two temp tables and unions them, puts the result in
temp_table1 and returns the result to the calling function
*****************************************************************/
union_tables_for_nested(temp_table1, temp_table)
char temp_table1[20];
char temp_table[20];
{

 int c=0,j=0,k=0,l=0,temp, count;
 /*char*/ STR_name char_value[21];
 char file_name[20],a;
 int  integer_value,media_value,found,media1_value;
 float real_value;
  int i=0,select=0;
int g=0;
/*printf("\nNow we are in union_tables_for_nested");*/
/* # line 3169 "db.sc" *//* select */
 {
   IIsqInit((char *)0);
   IIwritedb("retrieve(c=(count(");
   IIwritedb(temp_table1);
   IIwritedb(".");
   IIwritedb(satt[0].a_name);
   IIwritedb(")))");
   IIsqRinit((char *)0);
   if (IIerrtest() == 0) {
     if (IInextget() != 0) {
```

```
      IIretdom(1,30,4,&c);
     } /* IInextget */
     IIsqFlush((char *)0);
    } /* IIerrtest */
  }
 l=0;
 /*printf("\nThere are %d records in temp_table %s",c, temp_table1);*/

 /* # line 3171 "db.sc" *//* host code */
    if (IIcsrOpen((char *)0,"cursor_output","db1",0,temp_table1) != 0) {
     IIwritedb("retrieve(");
     for (select=0;select<n-1;select++) {
     IIwritedb(satt[select].a_name);
     IIwritedb("=");
     IIwritedb(temp_table1);
     IIwritedb(".");
     IIwritedb(satt[select].a_name);
     IIwritedb(",");
 }
     IIwritedb(satt[select].a_name);
     IIwritedb("=");
     IIwritedb(temp_table1);
     IIwritedb(".");
     IIwritedb(satt[select].a_name);
     IIwritedb(")");
     IIcsrQuery((char *)0);
    } /* IIcsrOpen */

 /*&&&&&&&&&&&&&&&&&&&&&&&&&&&*/
  {
    IIsqInit((char *)0);
    IIwritedb("create ");
    IIwritedb(temp_table);
    IIwritedb("(");
    for (i=0;i<n-1;i++){
      IIwritedb(satt[i].a_name);
  IIwritedb("=");
  if ((strcmp(satt[i].data_type, "image") == 0) ||
  (strcmp(satt[i].data_type, "sound") == 0) ||
     (strcmp(satt[i].data_type, "integer") == 0))
   IIwritedb("i4,");
  else
   if (strcmp(satt[i].data_type, "float") == 0)
     IIwritedb("f4,");
   else
     {                               /* char data_type */
```

```
    IIwritedb(satt[i].data_type);
    IIwritedb(",");
   }
   } /* End of for loop i */
  IIwritedb(satt[i].a_name);
  IIwritedb("=");
  if ((strcmp(satt[i].data_type, "image") == 0) ||
     (strcmp(satt[i].data_type, "sound") == 0) ||
          (strcmp(satt[i].data_type, "integer") == 0))
   IIwritedb("i4");
  else
   if (strcmp(satt[i].data_type, "float") == 0)
IIwritedb("f4");
   else
{                                 /* char data_type */
 IIwritedb(satt[i].data_type);
}
  IIwritedb(")");
  IIsqSync(0,(char *)0);
 }
/*&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&*/

/* # line 3169 "db.sc" *//* select */
 {
  IIsqInit((char *)0);
  IIwritedb("retrieve(g=(count(");
  IIwritedb(temp_table);
  IIwritedb(".");
  IIwritedb(satt[0].a_name);
  IIwritedb(")))");
  IIsqRinit((char *)0);
  if (IIerrtest() == 0) {
   if (IInextget() != 0) {
    IIretdom(1,30,4,&g);
   }
   IIsqFlush((char *)0);
  }
 }


/* # line 3171 "db.sc" *//* host code */
  if (IIcsrOpen((char *)0,"cursor_output","db2",0,temp_table) != 0) {
   IIwritedb("retrieve(");
   for (select=0;select<n-1;select++) {
   IIwritedb(satt[select].a_name);
   IIwritedb("=");
```

```
    IIwritedb(temp_table);
    IIwritedb(".");
    IIwritedb(satt[select].a_name);
    IIwritedb(",");
}
    IIwritedb(satt[select].a_name);
    IIwritedb("=");
    IIwritedb(temp_table);
    IIwritedb(".");
    IIwritedb(satt[select].a_name);
    IIwritedb(")");
    IIcsrQuery((char *)0);
   }

/*printf("\n");*/
look_more=0;
l=0;
if (c==0) {
look_more=1;
}

/* Fetch the cursor to the result relation which is the intermediate table
   hold the result from the query, then print out the tuple one at a time
   until no more record to print to the user */

/* # line 7 "insert.sc" *//* insert */
  {
   while (look_more == 0) {
    if (IIcsrFetch((char *)0,"cursor_output","db1") != 0) {
IIsqInit((char *)0);
IIwritedb("append to ");
IIwritedb(temp_table);
IIwritedb("(");
for (i=0;i<n-1;i++) {
 IIwritedb(satt[i].a_name);
 IIwritedb("=");
 if (strcmp(satt[i].data_type,"c20")==0) {
  IIcsrRet(1,32,0, char_value);
  IIsetdom(1,32,0, char_value);
 }
 if (strcmp(satt[i].data_type,"integer")==0) {
  IIcsrRet(1,30,4,&integer_value);
  IIsetdom(1,30,4,&integer_value);
 }
 if (strcmp(satt[i].data_type,"float")==0) {
  IIcsrRet(1,31,4,&real_value);
```

```
  IIsetdom(1,31,4,&real_value);
 }
 if (strcmp(satt[i].data_type,"image")==0) {
  IIcsrRet(1,30,4,&media_value);
  IIsetdom(1,30,4,&media_value);
 }
 if (strcmp(satt[i].data_type,"sound")==0) {
  IIcsrRet(1,30,4,&media1_value);
  IIsetdom(1,30,4,&media1_value);
 }
 IIwritedb(",");
}
IIwritedb(satt[i].a_name);
IIwritedb("=");
if (strcmp(satt[i].data_type,"c20")==0) {
 IIcsrRet(1,32,0, char_value);
 IIsetdom(1,32,0, char_value);
}
if (strcmp(satt[i].data_type,"integer")==0) {
 IIcsrRet(1,30,4,&integer_value);
 IIsetdom(1,30,4,&integer_value);
}
if (strcmp(satt[i].data_type,"float")==0) {
 IIcsrRet(1,31,4,&real_value);
 IIsetdom(1,31,4,&real_value);
}
if (strcmp(satt[i].data_type,"image")==0) {
 IIcsrRet(1,30,4,&media_value);
 IIsetdom(1,30,4,&media_value);
}
if (strcmp(satt[i].data_type,"sound")==0) {
 IIcsrRet(1,30,4,&media1_value);
 IIsetdom(1,30,4,&media1_value);
}

/*printf("\n");*/
IIcsrEFetch((char *)0);  /* fetch the next record to the cursor */
l++; /* increment l as the counter */
if (l==c) {   /* check if no more data to print */
 look_more =1; /* exit of the loop */
}
IIwritedb(" )");
IIsqSync(3,(char *)0);
     } /* IIcsrFetch */
   } /* end while */
  }
```

```
 IIcsrClose((char *)0,"cursor_output","db1"); /* close the cursor */
 IIcsrClose((char *)0,"cursor_output","db2"); /* close the cursor */
 return(temp_table);
}
/****************************************************************
This function takes two temp tables and unions them, puts the result in temp_table
and returns the result to the calling function
****************************************************************/
union_tables_for_demo(temp_table1, temp_table2, temp_table)
char temp_table1[20];
char temp_table2[20];
char temp_table[20];
{

 int c=0,j=0,k=0,l=0,temp, count;
 int o=0,p=0;
 /*char*/ STR_name char_value[21];
 char file_name[20],a;
 int  integer_value,media_value,found,media1_value;
 float real_value;
  int i=0,select=0;
int g=0;
/*printf("\nNow we are in union_tables_for_nested");*/
/* # line 3169 "db.sc" *//* select */
  {
   IIsqInit((char *)0);
   IIwritedb("retrieve(c=(count(");
   IIwritedb(temp_table1);
   IIwritedb(".");
   IIwritedb(satt[0].a_name);
   IIwritedb(")))");
   IIsqRinit((char *)0);
   if (IIerrtest() == 0) {
    if (IInextget() != 0) {
     IIretdom(1,30,4,&c);
    } /* IInextget */
    IIsqFlush((char *)0);
   } /* IIerrtest */
  }
l=0;
/*printf("\nThere are %d records in temp_table %s",c, temp_table1);*/

/* # line 3171 "db.sc" *//* host code */
   if (IIcsrOpen((char *)0,"cursor_output","db1",0,temp_table1) != 0) {
    IIwritedb("retrieve(");
    for (select=0;select<n-1;select++) {
```

```
    IIwritedb(satt[select].a_name);
    IIwritedb("=");
    IIwritedb(temp_table1);
    IIwritedb(".");
    IIwritedb(satt[select].a_name);
    IIwritedb(",");
}
    IIwritedb(satt[select].a_name);
    IIwritedb("=");
    IIwritedb(temp_table1);
    IIwritedb(".");
    IIwritedb(satt[select].a_name);
    IIwritedb(")");
    IIcsrQuery((char *)0);
   } /* IIcsrOpen */
/* # line 3169 "db.sc" *//* select */
 {
   IIsqInit((char *)0);
   IIwritedb("retrieve(o=(count(");
   IIwritedb(temp_table2);
   IIwritedb(".");
   IIwritedb(satt[0].a_name);
   IIwritedb(")))");
   IIsqRinit((char *)0);
   if (IIerrtest() == 0) {
    if (IInextget() != 0) {
     IIretdom(1,30,4,&o);
    } /* IInextget */
    IIsqFlush((char *)0);
   } /* IIerrtest */
 }
l=0;
/*printf("\nThere are %d records in temp_table %s",o, temp_table1);*/

/* # line 3171 "db.sc" *//* host code */
   if (IIcsrOpen((char *)0,"cursor_output","db3",0,temp_table2) != 0) {
    IIwritedb("retrieve(");
    for (select=0;select<n-1;select++) {
    IIwritedb(satt[select].a_name);
    IIwritedb("=");
    IIwritedb(temp_table2);
    IIwritedb(".");
    IIwritedb(satt[select].a_name);
    IIwritedb(",");
}
    IIwritedb(satt[select].a_name);
```

```
    IIwritedb("=");
    IIwritedb(temp_table2);
    IIwritedb(".");
    IIwritedb(satt[select].a_name);
    IIwritedb(")");
    IIcsrQuery((char *)0);
   } /* IIcsrOpen */

/*&&&&&&&&&&&&&&&&&&&&&&&&&&&*/
 {
  IIsqInit((char *)0);
  IIwritedb("create ");
  IIwritedb(temp_table);
  IIwritedb("(");
  for (i=0;i<n-1;i++){
    IIwritedb(satt[i].a_name);
 IIwritedb("=");
 if ((strcmp(satt[i].data_type, "image") == 0) ||
 (strcmp(satt[i].data_type, "sound") == 0) ||
    (strcmp(satt[i].data_type, "integer") == 0))
  IIwritedb("i4,");
 else
  if (strcmp(satt[i].data_type, "float") == 0)
   IIwritedb("f4,");
  else
   {                              /* char data_type */
    IIwritedb(satt[i].data_type);
    IIwritedb(",");
   }
    } /* End of for loop i */
  IIwritedb(satt[i].a_name);
  IIwritedb("=");
  if ((strcmp(satt[i].data_type, "image") == 0) ||
     (strcmp(satt[i].data_type, "sound") == 0) ||
          (strcmp(satt[i].data_type, "integer") == 0))
    IIwritedb("i4");
  else
    if (strcmp(satt[i].data_type, "float") == 0)
IIwritedb("f4");
    else
{                              /* char data_type */
 IIwritedb(satt[i].data_type);
}
  IIwritedb(")");
  IIsqSync(0,(char *)0);
 }
```

```
/*&&&&&&&&&&&&&&&&&&&&&&&&&&*/

/* # line 3169 "db.sc" *//* select */
 {
  IIsqInit((char *)0);
  IIwritedb("retrieve(g=(count(");
  IIwritedb(temp_table);
  IIwritedb(".");
  IIwritedb(satt[0].a_name);
  IIwritedb(")))");
  IIsqRinit((char *)0);
  if (IIerrtest() == 0) {
   if (IInextget() != 0) {
    IIretdom(1,30,4,&g);
   }
   IIsqFlush((char *)0);
  }
 }


/* # line 3171 "db.sc" *//* host code */
  if (IIcsrOpen((char *)0,"cursor_output","db2",0,temp_table) != 0) {
   IIwritedb("retrieve(");
   for (select=0;select<n-1;select++) {
   IIwritedb(satt[select].a_name);
   IIwritedb("=");
   IIwritedb(temp_table);
   IIwritedb(".");
   IIwritedb(satt[select].a_name);
   IIwritedb(",");
}
   IIwritedb(satt[select].a_name);
   IIwritedb("=");
   IIwritedb(temp_table);
   IIwritedb(".");
   IIwritedb(satt[select].a_name);
   IIwritedb(")");
   IIcsrQuery((char *)0);
  }

/*printf("\n");*/
look_more=0;
l=0;
if (c==0) {
look_more=1;
}
```

```
/* Fetch the cursor to the result relation which is the intermediate table
   hold the result from the query, then print out the tuple one at a time
   until no more record to print to the user */

/* # line 7 "insert.sc" *//* insert */
  {
   while (look_more == 0) {
     if (IIcsrFetch((char *)0,"cursor_output","db1") != 0) {
IIsqInit((char *)0);
IIwritedb("append to ");
IIwritedb(temp_table);
IIwritedb("(");
for (i=0;i<n-1;i++) {
 IIwritedb(satt[i].a_name);
 IIwritedb("=");
 if (strcmp(satt[i].data_type,"c20")==0) {
  IIcsrRet(1,32,0, char_value);
  IIsetdom(1,32,0, char_value);
 }
 if (strcmp(satt[i].data_type,"integer")==0) {
  IIcsrRet(1,30,4,&integer_value);
  IIsetdom(1,30,4,&integer_value);
 }
 if (strcmp(satt[i].data_type,"float")==0) {
  IIcsrRet(1,31,4,&real_value);
  IIsetdom(1,31,4,&real_value);
 }
 if (strcmp(satt[i].data_type,"image")==0) {
  IIcsrRet(1,30,4,&media_value);
  IIsetdom(1,30,4,&media_value);
 }
 if (strcmp(satt[i].data_type,"sound")==0) {
  IIcsrRet(1,30,4,&media1_value);
  IIsetdom(1,30,4,&media1_value);
 }
 IIwritedb(",");
}
IIwritedb(satt[i].a_name);
IIwritedb("=");
if (strcmp(satt[i].data_type,"c20")==0) {
 IIcsrRet(1,32,0, char_value);
 IIsetdom(1,32,0, char_value);
}
if (strcmp(satt[i].data_type,"integer")==0) {
 IIcsrRet(1,30,4,&integer_value);
```

```
 IIsetdom(1,30,4,&integer_value);
}
if (strcmp(satt[i].data_type,"float")==0) {
 IIcsrRet(1,31,4,&real_value);
 IIsetdom(1,31,4,&real_value);
}
if (strcmp(satt[i].data_type,"image")==0) {
 IIcsrRet(1,30,4,&media_value);
 IIsetdom(1,30,4,&media_value);
}
if (strcmp(satt[i].data_type,"sound")==0) {
 IIcsrRet(1,30,4,&media1_value);
 IIsetdom(1,30,4,&media1_value);
}
/*printf("\n");*/
IIcsrEFetch((char *)0);  /* fetch the next record to the cursor */
l++; /* increment l as the counter */

 if (l==c) {   /* check if no more data to print */
  look_more =1; /* exit of the loop */
 }
IIwritedb(" )");
IIsqSync(3,(char *)0);
     } /* IIcsrFetch */
   } /* end while */
 }
 IIcsrClose((char *)0,"cursor_output","db2"); /* close the cursor */


/* # line 3171 "db.sc" *//* host code */
   if (IIcsrOpen((char *)0,"cursor_output","db2",0,temp_table) != 0) {
     IIwritedb("retrieve(");
     for (select=0;select<n-1;select++) {
     IIwritedb(satt[select].a_name);
     IIwritedb("=");
     IIwritedb(temp_table);
     IIwritedb(".");
     IIwritedb(satt[select].a_name);
     IIwritedb(",");
 }
     IIwritedb(satt[select].a_name);
     IIwritedb("=");
     IIwritedb(temp_table);
     IIwritedb(".");
     IIwritedb(satt[select].a_name);
     IIwritedb(")");
```

```
    IIcsrQuery((char *)0);
   }

/*printf("\n");*/
look_more=0;
l=0;
if (c==0) {
look_more=1;
}

/* Fetch the cursor to the result relation which is the intermediate table
   hold the result from the query, then print out the tuple one at a time
   until no more record to print to the user */

/* # line 7 "insert.sc" *//* insert */
  {
   while (look_more == 0) {
     if (IIcsrFetch((char *)0,"cursor_output","db3") != 0) {
IIsqInit((char *)0);
IIwritedb("append to ");
IIwritedb(temp_table);
IIwritedb("(");
for (i=0;i<n-1;i++) {
  IIwritedb(satt[i].a_name);
  IIwritedb("=");
  if (strcmp(satt[i].data_type,"c20")==0) {
   IIcsrRet(1,32,0, char_value);
   IIsetdom(1,32,0, char_value);
  }
  if (strcmp(satt[i].data_type,"integer")==0) {
   IIcsrRet(1,30,4,&integer_value);
   IIsetdom(1,30,4,&integer_value);
  }
  if (strcmp(satt[i].data_type,"float")==0) {
   IIcsrRet(1,31,4,&real_value);
   IIsetdom(1,31,4,&real_value);
  }
  if (strcmp(satt[i].data_type,"image")==0) {
   IIcsrRet(1,30,4,&media_value);
   IIsetdom(1,30,4,&media_value);
  }
  if (strcmp(satt[i].data_type,"sound")==0) {
   IIcsrRet(1,30,4,&media1_value);
   IIsetdom(1,30,4,&media1_value);
  }
  IIwritedb(",");
```

```
}
IIwritedb(satt[i].a_name);
IIwritedb("=");
if (strcmp(satt[i].data_type,"c20")==0) {
 IIcsrRet(1,32,0, char_value);
 IIsetdom(1,32,0, char_value);
}
if (strcmp(satt[i].data_type,"integer")==0) {
 IIcsrRet(1,30,4,&integer_value);
 IIsetdom(1,30,4,&integer_value);
}
if (strcmp(satt[i].data_type,"float")==0) {
 IIcsrRet(1,31,4,&real_value);
 IIsetdom(1,31,4,&real_value);
}
if (strcmp(satt[i].data_type,"image")==0) {
 IIcsrRet(1,30,4,&media_value);
 IIsetdom(1,30,4,&media_value);
}
if (strcmp(satt[i].data_type,"sound")==0) {
 IIcsrRet(1,30,4,&media1_value);
 IIsetdom(1,30,4,&media1_value);
}
/*printf("\n");*/
IIcsrEFetch((char *)0);  /* fetch the next record to the cursor */
l++; /* increment l as the counter */

  if (l==o) {   /* check if no more data to print */
   look_more =1; /* exit of the loop */
  }
IIwritedb(" )");
IIsqSync(3,(char *)0);
     } /* IIcsrFetch */
    } /* end while */
  }

  IIcsrClose((char *)0,"cursor_output","db2"); /* close the cursor */
  IIcsrClose((char *)0,"cursor_output","db1"); /* close the cursor */
  IIcsrClose((char *)0,"cursor_output","db3"); /* close the cursor */
  return(temp_table);
}
/*****************************************************************
This function retrieves the tuples from temp_table1 which do not take place in
temp_table2 and puts the result in temp_table.
*****************************************************************/
minus(temp_table1, temp_table2, temp_table)
```

```
char temp_table1[20];
char temp_table2[20];
char temp_table[20];
{
 int i;
IIsqInit((char *)0);
IIwritedb("retrieve into ");
IIwritedb(temp_table);
IIwritedb("(");
for (i=0;i<n-1;i++) {
 IIwritedb(temp_table1);
 IIwritedb(".");
 IIwritedb(satt[i].a_name);
 IIwritedb(",");
}
IIwritedb(temp_table1);
IIwritedb(".");
IIwritedb(satt[i].a_name);
    IIwritedb(")");

 IIwritedb("where any(");
 IIwritedb(temp_table2);
 IIwritedb(".all by ");
 IIwritedb(temp_table1);
 IIwritedb(".all ");
 IIwritedb(" where(");
 for (i=0;i<n-1;i++) {
  IIwritedb(temp_table1);
  IIwritedb(".");
  IIwritedb(satt[i].a_name);
  IIwritedb("=");
  IIwritedb(temp_table2);
  IIwritedb(".");
  IIwritedb(satt[i].a_name);
  IIwritedb(" and ");
 }
 IIwritedb(temp_table1);
 IIwritedb(".");
 IIwritedb(satt[i].a_name);
 IIwritedb("=");
 IIwritedb(temp_table2);
 IIwritedb(".");
 IIwritedb(satt[i].a_name);
 IIwritedb(" )");
 IIwritedb(") = 0");
 IIsqSync(0,(char *)0);
```

```
 return(temp_table);
}
/*****************************************************************
This function intersects two tables (1 and 2) and puts the result in temp_table.
*****************************************************************/
intersect_tables(temp_table1, temp_table2, temp_table)
char temp_table1[20];
char temp_table[20];
{
 int i;
/*copy_to_file(temp_table1);*/
 IIsqInit((char *)0);
IIwritedb("retrieve into ");
IIwritedb(temp_table);
IIwritedb("(");
for (i=0;i<n-1;i++) {
 IIwritedb(temp_table1);
 IIwritedb(".");
 IIwritedb(satt[i].a_name);
 IIwritedb(",");
}
IIwritedb(temp_table1);
IIwritedb(".");
IIwritedb(satt[i].a_name);
    IIwritedb(")");

    IIwritedb(" where(");
for (i=0;i<n-1;i++) {
IIwritedb(temp_table1);
IIwritedb(".");
IIwritedb(satt[i].a_name);
IIwritedb("=");
IIwritedb(temp_table2);
IIwritedb(".");
IIwritedb(satt[i].a_name);
IIwritedb(" and ");
   }
IIwritedb(temp_table1);
IIwritedb(".");
IIwritedb(satt[i].a_name);
IIwritedb("=");
IIwritedb(temp_table2);
IIwritedb(".");
IIwritedb(satt[i].a_name);
IIwritedb(" )");
IIsqSync(0,(char *)0);
```

```
     return(temp_table);
}
/***********************************************************
This function retrieves the tuples from temp_table1 which are not included in
temp2 and puts the result in temp_table.
***********************************************************/
temp1_not_in_temp2(temp_table1, temp_table2, temp_table)
char temp_table[20];
char temp_table1[20];
char temp_table2[20];
{
  int i,j;
  j=0;

  printf("\nWe are in table1_NOT_IN_table2 now");

  sqlca.sqlcode = 0;  /* Initialize as error free before access INGRES */
  IIsqInit(&sqlca);
  IIwritedb("retrieve into ");
  IIwritedb(temp_table);
  IIwritedb("(");
  for (i=0;i<n-1;i++) {
    IIwritedb(satt[i].t_name);
    IIwritedb(".");
    IIwritedb(satt[i].a_name);
    IIwritedb(",");
  }
  IIwritedb(satt[i].t_name);
  IIwritedb(".");
  IIwritedb(satt[i].a_name);
  IIwritedb(")");

  IIwritedb("where(any(");
  IIwritedb(temp_table2);
  IIwritedb(".");
  IIwritedb(attribute_for_nested);
/*  IIwritedb(".all by ");*/
/*  IIwritedb(temp_table1);*/
  IIwritedb(" by ");
  IIwritedb(satt[i].t_name);
  IIwritedb(".all ");
  IIwritedb("where ");
  IIwritedb("(");
  IIwritedb(satt[i].t_name);
/*  IIwritedb(temp_table1);*/
  IIwritedb(".");
```

```
IIwritedb(condition_for_nested);
IIwritedb("=");
IIwritedb(temp_table2);
IIwritedb(".");
IIwritedb(attribute_for_nested);
IIwritedb(")");
IIwritedb(") = 0");
if (m>1){
 IIwritedb(" and ");
 IIwritedb("(");
 IIwritedb(join_condition);
 IIwritedb(")");
 }
IIwritedb(")");
IIsqSync(0,&sqlca);

if (sqlca.sqlcode != 0){
 printf("\nAn error occurred while accessing the database");
 for (j=j+1; j<m; j++){
  init_buffer(temp_table1,20);
  strcpy(temp_table1, stab[j].t_name);

  sqlca.sqlcode = 0;  /* Initialize as error free before access INGRES */

  IIsqInit(&sqlca);
  IIwritedb("retrieve into ");
  IIwritedb(temp_table);
  IIwritedb("(");
  for (i=0;i<n-1;i++) {
IIwritedb(satt[i].t_name);
IIwritedb(".");
IIwritedb(satt[i].a_name);
IIwritedb(",");
  }
  IIwritedb(satt[i].t_name);
  IIwritedb(".");
  IIwritedb(satt[i].a_name);
  IIwritedb(")");

  IIwritedb("where(any(");
  IIwritedb(temp_table2);
  IIwritedb(".");
  IIwritedb(attribute_for_nested);
  IIwritedb(" by ");
/*   IIwritedb(".all by ");*/
  IIwritedb(satt[i].t_name);
```

```
/*    IIwritedb(temp_table1);*/
    IIwritedb(".all ");
    IIwritedb("where ");
    IIwritedb("(");
    IIwritedb(satt[i].t_name);
/*    IIwritedb(temp_table1);*/
    IIwritedb(".");
    IIwritedb(condition_for_nested);
    IIwritedb("=");
    IIwritedb(temp_table2);
    IIwritedb(".");
    IIwritedb(attribute_for_nested);
    IIwritedb(")");
    IIwritedb(") = 0");
    if (m>1){
IIwritedb(" and ");
IIwritedb("(");
IIwritedb(join_condition);
IIwritedb(")");
    }
    IIwritedb(")");
    IIsqSync(0,&sqlca);

   }/* end for j<m */
  }/* end if */

 }
/************************************************************
This function joins temp1 and temp2 and retrives the tuples from temp1 that takes place in temp2
and puts the result in temp_table.
************************************************************/
temp1_in_temp2(temp_table1, temp_table2, temp_table)
char temp_table[20];
char temp_table1[20];
char temp_table2[20];
{
  int i,j;
  j=0;

  printf("\nWe are in table1_IN_table2 now");
  sqlca.sqlcode = 0;  /* Initialize as error free before access INGRES */
  IIsqInit(&sqlca);
  IIwritedb("retrieve into ");
  IIwritedb(temp_table);
  IIwritedb("(");
  for (i=0;i<n-1;i++) {
```

```
  IIwritedb(satt[i].t_name);
  IIwritedb(".");
  IIwritedb(satt[i].a_name);
  IIwritedb(",");
 }
 IIwritedb(satt[i].t_name);
 IIwritedb(".");
 IIwritedb(satt[i].a_name);
 IIwritedb(")");

 IIwritedb("where(");
 IIwritedb("(");
 IIwritedb(temp_table1);
 IIwritedb(".");
 IIwritedb(condition_for_nested);
 IIwritedb("=");
 IIwritedb(temp_table2);
 IIwritedb(".");
 IIwritedb(attribute_for_nested);
 IIwritedb(")");
 if (m>1){
  IIwritedb(" and ");
  IIwritedb("(");
  IIwritedb(join_condition);
  IIwritedb(")");
 }
 IIwritedb(")");
 IIsqSync(0,&sqlca);

 if (sqlca.sqlcode != 0){
  for (j=j+1; j<m; j++){
   init_buffer(temp_table1,20);
   strcpy(temp_table1, stab[j].t_name);
   sqlca.sqlcode = 0; /* Initialize as error free before access INGRES */
   IIsqInit(&sqlca);
   IIwritedb("retrieve into ");
   IIwritedb(temp_table);
   IIwritedb("(");
   for (i=0;i<n-1;i++) {
IIwritedb(satt[i].t_name);
IIwritedb(".");
IIwritedb(satt[i].a_name);
IIwritedb(",");
   }
   IIwritedb(satt[i].t_name);
   IIwritedb(".");
```

```
    IIwritedb(satt[i].a_name);
    IIwritedb(")");

    IIwritedb("where(");
    IIwritedb("(");
    IIwritedb(temp_table1);
    IIwritedb(".");
    IIwritedb(condition_for_nested);
    IIwritedb("=");
    IIwritedb(temp_table2);
    IIwritedb(".");
    IIwritedb(attribute_for_nested);
    IIwritedb(")");
    if (m>1){
IIwritedb(" and ");
IIwritedb("(");
IIwritedb(join_condition);
IIwritedb(")");
     }
     IIwritedb(")");
     IIsqSync(0,&sqlca);
   }/* end for */
  }/* end if */

}
/************************************************************
This function joins temp1 and temp2 and retrieves the tuples from temp1 that do not take place in
temp2 and puts the result in temp_table.
************************************************************/
temp1_not_exists_temp2(temp_table1, temp_table2, temp_table)
char temp_table[20];
char temp_table1[20];
char temp_table2[20];
{
 int i,j;
 j=0;
 printf("\nWe are in table1_not_exists_table2 now");
 sqlca.sqlcode = 0; /* Initialize as error free before access INGRES */

 IIsqInit(&sqlca);
 IIwritedb("retrieve into ");
 IIwritedb(temp_table);
 IIwritedb("(");
 for (i=0;i<n-1;i++) {
  IIwritedb(satt[i].t_name);
  IIwritedb(".");
```

```
    IIwritedb(satt[i].a_name);
    IIwritedb(",");
  }
  IIwritedb(satt[i].t_name);
  IIwritedb(".");
  IIwritedb(satt[i].a_name);
  IIwritedb(")");

  IIwritedb("where(any(");
  IIwritedb(temp_table2);
  IIwritedb(".all by ");
  IIwritedb(satt[i].t_name);
/* IIwritedb(temp_table1);*/
  IIwritedb(".all ");
  IIwritedb("where ");
  IIwritedb("(");
  IIwritedb(join_for_nested);
  IIwritedb(")");
  IIwritedb(")=0");
  if (m>1){
    IIwritedb(" and ");
    IIwritedb("(");
    IIwritedb(join_condition);
    IIwritedb(")");
  }
  IIwritedb(")");

  IIsqSync(0,&sqlca);

  if (sqlca.sqlcode != 0){
    printf("\nError occurred while accessing the database");
    for (j=j+1; j<m; j++){
      init_buffer(temp_table1,20);
      strcpy(temp_table1, stab[j].t_name);
      sqlca.sqlcode = 0;  /* Initialize as error free before access INGRES */

      IIsqInit(&sqlca);
      IIwritedb(temp_table);
      IIwritedb("(");
      for (i=0;i<n-1;i++) {
IIwritedb(satt[i].t_name);
IIwritedb(".");
IIwritedb(satt[i].a_name);
IIwritedb(",");
      }
      IIwritedb(satt[i].t_name);
```

```
    IIwritedb(".");
    IIwritedb(satt[i].a_name);
    IIwritedb(")");

    IIwritedb("where(any(");
    IIwritedb(temp_table2);
    IIwritedb(".all by ");
    IIwritedb(satt[i].t_name);
    /*IIwritedb(temp_table1);*/
    IIwritedb(".all ");
    IIwritedb("where ");
    IIwritedb("(");
    IIwritedb(join_for_nested);
    IIwritedb(")");
    IIwritedb(")=0");
    if (m>1){
IIwritedb(" and ");
IIwritedb("(");
IIwritedb(join_condition);
IIwritedb(")");
     }
     IIwritedb(")");
     IIsqSync(0,&sqlca);

   }/* end j<m */
  }/* end sqlca.sqlcode != 0 */

}
/************************************************************
This function retrives the tuples from temp1 that exists in temp2 and puts the
result in temp_table.
************************************************************/
temp1_exists_temp2(temp_table1, temp_table2, temp_table)
char temp_table[20];
char temp_table1[20];
char temp_table2[20];
{
  int i,j;
   j=0;

  printf("\nWe are in table1_exists_table2 now");
  sqlca.sqlcode = 0; /* Initialize as error free before access INGRES */

  IIsqInit(&sqlca);
  IIwritedb("retrieve into ");
  IIwritedb(temp_table);
```

```
IIwritedb("(");
for (i=0;i<n-1;i++) {
  IIwritedb(satt[i].t_name);
  IIwritedb(".");
  IIwritedb(satt[i].a_name);
  IIwritedb(",");
}
IIwritedb(satt[i].t_name);
IIwritedb(".");
IIwritedb(satt[i].a_name);
IIwritedb(")");

IIwritedb("where(");
IIwritedb("(");
IIwritedb(join_for_nested);
IIwritedb(")");
if (m>1){
  IIwritedb(" and ");
  IIwritedb("(");
  IIwritedb(join_condition);
  IIwritedb(")");
}
IIwritedb(")");
IIsqSync(0,&sqlca);

if (sqlca.sqlcode != 0){
  for (j=j+1; j<m; j++){
    init_buffer(temp_table1,20);
    strcpy(temp_table1, stab[j].t_name);
    sqlca.sqlcode = 0;  /* Initialize as error free before access INGRES */

    IIsqInit(&sqlca);
    IIwritedb("retrieve into ");
    IIwritedb(temp_table);
    IIwritedb("(");
    for (i=0;i<n-1;i++) {
IIwritedb(satt[i].t_name);
IIwritedb(".");
IIwritedb(satt[i].a_name);
IIwritedb(",");
    }
    IIwritedb(satt[i].t_name);
    IIwritedb(".");
    IIwritedb(satt[i].a_name);
    IIwritedb(")");
```

```
    IIwritedb("where(");
    IIwritedb("(");
    IIwritedb(join_for_nested);
    IIwritedb(")");
    if (m>1){
IIwritedb(" and ");
IIwritedb("(");
IIwritedb(join_condition);
IIwritedb(")");
    }
    IIwritedb(")");
    IIsqSync(0,&sqlca);
   }/* end if j<m */
  }/* end for */
}
/**********************************************
This function calculates the number of tuples retrieved in the result table and prints the number of tuples.
**********************************************/
void print_count(temp_table, i)
char temp_table[20];
int i;
{
 int t=0;
   {
    IIsqInit((char *)0);
    IIwritedb("retrieve unique(t=(");
    IIwritedb("count");
    IIwritedb("(");
    IIwritedb(temp_table);
    IIwritedb(".");
    IIwritedb(satt[i].a_name);
    IIwritedb(")))");
    IIsqRinit((char *)0);
    if (IIerrtest() == 0) {
if (IInextget() != 0) {
 IIretdom(1,30,4,&t);
} /* IInextget */
IIsqFlush((char *)0);
    } /* IIerrtest */
   }
 printf("COUNT(%s) = %d  ",satt[i].a_name, t);
}
/**********************************************
This function calculates the sum of a cloumn retrieved in the result table and prints the sum.
**********************************************/
```

```
void print_sum(temp_table, i)
char temp_table[20];
int i;
{
 int t=0;
  {
    IIsqInit((char *)0);
    IIwritedb("retrieve unique(t=(");
    IIwritedb("sum");
    IIwritedb("(");
    IIwritedb(temp_table);
    IIwritedb(".");
    IIwritedb(satt[i].a_name);
    IIwritedb(")))");
    IIsqRinit((char *)0);
    if (IIerrtest() == 0) {
if (IInextget() != 0) {
 IIretdom(1,30,4,&t);
} /* IInextget */
IIsqFlush((char *)0);
    } /* IIerrtest */
  }
 printf("SUM(%s) = %d  ",satt[i].a_name, t);
}
/***********************************************
This function calculates the average of an attribute of a tuple retrieved in the result table and
prints the average.
***********************************************/
void print_avg(temp_table, i)
char temp_table[20];
int i;
{
 int t=0;
  {
    IIsqInit((char *)0);
    IIwritedb("retrieve unique(t=(");
    IIwritedb("avg");
    IIwritedb("(");
    IIwritedb(temp_table);
    IIwritedb(".");
    IIwritedb(satt[i].a_name);
    IIwritedb(")))");
    IIsqRinit((char *)0);
    if (IIerrtest() == 0) {
if (IInextget() != 0) {
 IIretdom(1,30,4,&t);
```

```
} /* IInextget */
IIsqFlush((char *)0);
    } /* IIerrtest */
   }
 printf("AVG(%s) = %d",satt[i].a_name, t);
}
/**********************************************
This function finds the max of a coloumn of a tuple in the temp_table and prints the max.
**********************************************/
void print_max(temp_table, i)
char temp_table[20];
int i;
{
 int t=0;
   {
    IIsqInit((char *)0);
    IIwritedb("retrieve unique(t=(");
    IIwritedb("max");
    IIwritedb("(");
    IIwritedb(temp_table);
    IIwritedb(".");
    IIwritedb(satt[i].a_name);
    IIwritedb(")))");
    IIsqRinit((char *)0);
    if (IIerrtest() == 0) {
if (IInextget() != 0) {
 IIretdom(1,30,4,&t);
} /* IInextget */
IIsqFlush((char *)0);
    } /* IIerrtest */
   }
 printf("MAX(%s) = %d  ",satt[i].a_name, t);
}
/**********************************************
This function calculates the min of an attribute of a tuple retrieved in the temp_table and prints the
min.
**********************************************/
void print_min(temp_table, i)
char temp_table[20];
int i;
{
 int t=0;
   {
    IIsqInit((char *)0);
    IIwritedb("retrieve unique(t=(");
    IIwritedb("min");
```

```
    IIwritedb("(");
    IIwritedb(temp_table);
    IIwritedb(".");
    IIwritedb(satt[i].a_name);
    IIwritedb(")))");
    IIsqRinit((char *)0);
    if (IIerrtest() == 0) {
if (IInextget() != 0) {
 IIretdom(1,30,4,&t);
} /* IInextget */
IIsqFlush((char *)0);
    } /* IIerrtest */
   }
 printf("MIN(%s) = %d  ",satt[i].a_name, t);
}
/*****************************************************************
This function checks the aggregate type in the struct satt and calls the appropriate function.
*****************************************************************/
print_aggregates(temp_table)
char temp_table[20];
{
 int v;
 for(v=0; v<n; v++){
  if (satt[v].aggregate_type==1)
   print_count(temp_table, v);
  if (satt[v].aggregate_type==2)
   print_sum(temp_table, v);
  if (satt[v].aggregate_type==3)
   print_avg(temp_table, v);
  if (satt[v].aggregate_type==4)
   print_max(temp_table, v);
  if (satt[v].aggregate_type==5)
   print_min(temp_table, v);
 }
}
/*****************************************************************
This function prints the tuples retrived in the result table.
*****************************************************************/
print_result_table(temp_table,flag,c)
char temp_table[20];
int flag;
int c;
{
 int v;
 int j=0,k=0,l=0,temp,select=0;
 char char_value[21],a, Ans;
```

```
  char file_name[20];
  int  integer_value,media_value,found,media1_value;
  float real_value;
  int record_id;
  int i=0;
  c=0;
/* # line 3169 "db.sc" *//* select */
  {
    IIsqInit((char *)0);
    IIwritedb("retrieve unique(c=(count(");
    IIwritedb(temp_table);
    IIwritedb(".");
    IIwritedb(satt[0].a_name);
    IIwritedb(")))");
    IIsqRinit((char *)0);
    if (IIerrtest() == 0) {
      if (IInextget() != 0) {
        IIretdom(1,30,4,&c);
      } /* IInextget */
      IIsqFlush((char *)0);
    } /* IIerrtest */
  }
 l=0;
 if (flag==FALSE){
  printf("\nThere are %d records that match the query",c);
  /*--------*/
  if (c==0) {
    printf("\nPress ENTER to continue...");
    a=getchar();
    return;
  }
 }
/* # line 3171 "db.sc" *//* host code */
    if (IIcsrOpen((char *)0,"cursor_output","db1",0,temp_table) != 0) {
      IIwritedb("retrieve (");
      for (select=0;select<n-1;select++) {
      IIwritedb(satt[select].a_name);
      IIwritedb("=");
      IIwritedb(temp_table);
      IIwritedb(".");
      IIwritedb(satt[select].a_name);
      IIwritedb(",");
 }
      IIwritedb(satt[select].a_name);
      IIwritedb("=");
      IIwritedb(temp_table);
```

```
    IIwritedb(".");
    IIwritedb(satt[select].a_name);
    IIwritedb(")");
    IIcsrQuery((char *)0);
  } /* IIcsrOpen */
printf("\n");
look_more=0;
l=0;
if (c==0) {
look_more=1;
}
/* Fetch the cursor to the result relation which is the intermediate table
  hold the result from the query, then print out the tuple one at a time
  until no more record to print to the user */
while (look_more == 0) {
  if (IIcsrFetch((char *)0,"cursor_output","db1") != 0) {
printf("record id %d \t",l+1);
     for (i=0;i<n;i++) {
  if (strcmp(satt[i].data_type,"c20")==0) {
     IIcsrRet(1,32,0,char_value);
if (satt[i].aggregate_type==0)
 printf("%s : %s",satt[i].a_name,char_value);

}
  if (strcmp(satt[i].data_type,"integer")==0) {
     IIcsrRet(1,30,4,&integer_value);
if (satt[i].aggregate_type==0)
 printf("%s : %d  ",satt[i].a_name,integer_value);
}
  if (strcmp(satt[i].data_type,"float")==0) {
     IIcsrRet(1,31,4,&real_value);
if (satt[i].aggregate_type==0)
 printf("%s : %8.2f  ",satt[i].a_name,real_value);
}
  if (strcmp(satt[i].data_type,"image")==0) {
     IIcsrRet(1,30,4,&media_value);
if (satt[i].aggregate_type==0)
 printf("%s id is %d  ",satt[i].a_name,media_value);
    }
  if (strcmp(satt[i].data_type,"sound")==0) {
     IIcsrRet(1,30,4,&media1_value);
if (satt[i].aggregate_type==0)
 printf("%s id is %d",satt[i].a_name,media1_value);
}
 } /* end for select < n*/
IIcsrEFetch((char *)0);  /* fetch the next record to the cursor */
```

```
l++; /* increment l as the counter */
if (l==c) {   /* check if no more data to print */
 look_more =1;  /* exit of the loop */
}
print_aggregates(temp_table);
printf("\n");
     } /* IIcsrFetch */
  } /* end while */
 IIcsrClose((char *)0,"cursor_output","db1"); /* close the cursor */
 if (flag==FALSE){
   printf("Press ENTER to continue ..");
   a= getchar();
  }
/* this for the check for the media selection */
 if (c==0) {
i=9999;
}

/* if there are some aggregate functions print their results */
/*  print_aggregates(temp_table);
 printf("\nPress ENTER to continue ..");
 a= getchar();*/
 return(c);
}
/*****************************************************************
This function gets the image id of a tuple in the result table so it can locate the image in
the image table associated with the result table for the delete and modify procedures.
*****************************************************************/
get_image_id(r,image_id)
int r;
int image_id;
{
  int sound_id;
  int entry;
  int desired_tuple;
  char c_temp[60];
  int count=0;
  int j=0,k=0,l=0,temp;
  char char_value[21],a;
  char file_name[20];
  int img_value, snd_value;
  int  integer_value,media_value,found,media1_value;
  float real_value;
   int i=0,select=0;
  int g=0;
  int d;
```

```
 i_value[i_index]=0;
 desired_tuple=r;
/* # line 3169 "db.sc" *//* select */
 {
  IIsqInit((char *)0);
  IIwritedb("retrieve(g=(count(");
  IIwritedb(temp_table);
  IIwritedb(".");
  IIwritedb(satt[0].a_name);
  IIwritedb(")))");
  IIsqRinit((char *)0);
  if (IIerrtest() == 0) {
   if (IInextget() != 0) {
    IIretdom(1,30,4,&g);
   } /* IInextget */
   IIsqFlush((char *)0);
  } /* IIerrtest */
 }
l=0;
if (g==0) {
printf("\nPress ENTER to continue...");
a=getchar();
return;
}
/* # line 3171 "db.sc" *//* host code */
  if (IIcsrOpen((char *)0,"cursor_output","db2",0,temp_table) != 0) {
   IIwritedb("retrieve(");
   for (select=0;select<n-1;select++) {
   IIwritedb(satt[select].a_name);
   IIwritedb("=");
   IIwritedb(temp_table);
   IIwritedb(".");
   IIwritedb(satt[select].a_name);
   IIwritedb(",");
}
   IIwritedb(satt[select].a_name);
   IIwritedb("=");
   IIwritedb(temp_table);
   IIwritedb(".");
   IIwritedb(satt[select].a_name);
   IIwritedb(")");
   IIcsrQuery((char *)0);
  } /* IIcsrOpen */
 printf("\n");
 look_more=0;
 l=0;
```

```
 if (g==0) {
  look_more=1;
 }
 /* Fetch the cursor to the result relation which is the intermediate table
    hold the result from the query, then print out the tuple one at a time
    until no more record to print to the user */

 while (look_more == 0) {
  if (IIcsrFetch((char *)0,"cursor_output","db2") != 0) {
   if (desired_tuple == 1){
printf("record id %d \t",l+1);
    }
for (i=0;i<n;i++) {
 if (strcmp(satt[i].data_type,"c20")==0) {
  IIcsrRet(1,32,0,char_value);
  if (desired_tuple == 1){
   printf("%s :  %s",satt[i].a_name,char_value);
  }
 }
 if (strcmp(satt[i].data_type,"integer")==0) {
  IIcsrRet(1,30,4,&integer_value);
  if (desired_tuple == 1){
   printf("%s :  %d  ",satt[i].a_name,integer_value);
  }
 }
 if (strcmp(satt[i].data_type,"float")==0) {
  IIcsrRet(1,31,4,&real_value);
  if (desired_tuple == 1){
   printf("%s :  %8.2f  ",satt[i].a_name,real_value);
  }
 }
 if (strcmp(satt[i].data_type,"image")==0) {
  IIcsrRet(1,30,4,&media_value);
  if (desired_tuple == 1){
   image_id=media_value;
   printf("%s id is %d  ",satt[i].a_name,media_value);
  }
 }
 if (strcmp(satt[i].data_type,"sound")==0) {
  IIcsrRet(1,30,4,&media1_value);
  if (desired_tuple == 1){
/*     sound_id=media1_value;*/
   printf("%s %d",satt[i].a_name,media1_value);
  }
 }
}/*end for select n*/
```

```
    IIcsrEFetch((char *)0);  /* fetch the next record to the cursor */
    l++; /* increment l as the counter */
    if (l==g) {  /* check if no more data to print */
look_more =1; /* exit of the loop */
    }
   } /* IIcsrFetch */
  } /* end while */

  IIcsrClose((char *)0,"cursor_output","db2"); /* close the cursor */

  printf("\nPress ENTER to continue ..");
  a= getchar();
  return(image_id);
}
/************************************************************************
This function gets the sound id of a tuple in the result tableto find the associated sound data
and delete it. Used in delete and modify operations.
************************************************************************/
get_sound_id(r,sound_id)
int r;
int sound_id;
{
  int image_id;
  int entry;
  int desired_tuple;
  char c_temp[60];
  int count=0;
  int j=0,k=0,l=0,temp;
  char char_value[21],a;
  char file_name[20];
  int img_value, snd_value;
  int  integer_value,media_value,found,media1_value;
  float real_value;
   int i=0,select=0;
  int g=0;
  int d;
  i_value[i_index]=0;
  desired_tuple=r;
/* # line 3169 "db.sc" *//* select */
  {
    IIsqInit((char *)0);
    IIwritedb("retrieve(g=(count(");
    IIwritedb(temp_table);
    IIwritedb(".");
    IIwritedb(satt[0].a_name);
    IIwritedb(")))");
```

```
    IIsqRinit((char *)0);
    if (IIerrtest() == 0) {
      if (IInextget() != 0) {
        IIretdom(1,30,4,&g);
      } /* IInextget */
      IIsqFlush((char *)0);
    } /* IIerrtest */
  }
l=0;
if (g==0) {
printf("\nPress ENTER to continue...");
a=getchar();
return;
}
/* # line 3171 "db.sc" *//* host code */
    if (IIcsrOpen((char *)0,"cursor_output","db2",0,temp_table) != 0) {
      IIwritedb("retrieve(");
      for (select=0;select<n-1;select++) {
      IIwritedb(satt[select].a_name);
      IIwritedb("=");
      IIwritedb(temp_table);
      IIwritedb(".");
      IIwritedb(satt[select].a_name);
      IIwritedb(",");
}
      IIwritedb(satt[select].a_name);
      IIwritedb("=");
      IIwritedb(temp_table);
      IIwritedb(".");
      IIwritedb(satt[select].a_name);
      IIwritedb(")");
      IIcsrQuery((char *)0);
    } /* IIcsrOpen */
  printf("\n");
  look_more=0;
  l=0;
  if (g==0) {
    look_more=1;
  }
  /* Fetch the cursor to the result relation which is the intermediate table
     hold the result from the query, then print out the tuple one at a time
     until no more record to print to the user */

  while (look_more == 0) {
    if (IIcsrFetch((char *)0,"cursor_output","db2") != 0) {
      if (desired_tuple == 1){
```

```
printf("record id %d \t",l+1);
   }
for (i=0;i<n;i++) {
 if (strcmp(satt[i].data_type,"c20")==0) {
  IIcsrRet(1,32,0,char_value);
  if (desired_tuple == l){
   printf("%s :  %s",satt[i].a_name,char_value);
  }
 }
 if (strcmp(satt[i].data_type,"integer")==0) {
  IIcsrRet(1,30,4,&integer_value);
  if (desired_tuple == l){
   printf("%s :  %d  ",satt[i].a_name,integer_value);
  }
 }
 if (strcmp(satt[i].data_type,"float")==0) {
  IIcsrRet(1,31,4,&real_value);
  if (desired_tuple == l){
   printf("%s :  %8.2f  ",satt[i].a_name,real_value);
  }
 }
 if (strcmp(satt[i].data_type,"image")==0) {
  IIcsrRet(1,30,4,&media_value);
  if (desired_tuple == l){
/*     image_id=media_value;*/
   printf("%s id is %d  ",satt[i].a_name,media_value);
  }
 }
 if (strcmp(satt[i].data_type,"sound")==0) {
  IIcsrRet(1,30,4,&media1_value);
  if (desired_tuple == l){
   sound_id=media1_value;
   printf("%s %d",satt[i].a_name,media1_value);
  }
 }
}/*end for select n*/
   IIcsrEFetch((char *)0);  /* fetch the next record to the cursor */
   l++; /* increment l as the counter */
   if (l==g) {  /* check if no more data to print */
lock_more =1; /* exit of the loop */
   }
  } /* IIcsrFetch */
 } /* end while */

 IIcsrClose((char *)0,"cursor_output","db2"); /* close the cursor */
 return(sound_id);
```

```
}

/*****************************************************************
This function calls the function print_result_table and then queries the user if he wants to display
any media data.
*****************************************************************/
ql_printdata(temp_table)
char temp_table[20];

{
 int image_id=0;
 int sound_id=0;
 int c=0,j=0,k=0,l=0,temp,select=0;
 char char_value[21],a, Ans;
 char file_name[20];
 int  integer_value,media_value,found,media1_value;
 float real_value;
 int record_id, flag=FALSE;
 int i=0;
 c=print_result_table(temp_table, flag, c);
 flag=TRUE;

for (k=0;k<n;k++) {
 if ((strcmp(satt[k].data_type,"image")==0)||(strcmp(satt[k].data_type,"sound")==0)) {
  if (strcmp(satt[k].data_type,"image")==0)
   printf("\nDo you want to display any media data ? (y/n)");
  if (strcmp(satt[k].data_type,"sound")==0)
   printf("\nDo you want to display any media data ? (y/n)");
  Ans=yes_no_answer();
  if ((Ans==121)||(Ans==89)){
 for (k=0;k<n;k++) {
  if (strcmp(satt[k].data_type,"image")==0) {
 Ans =121;
 while ((Ans == 121) || (Ans == 89)){
  if (c>1){
   printf("\n\nWhich tuple's image do you want to see? (enter record id) :");
   scanf("%d", &record_id);
   getchar();
   printf("record_id --> %d", record_id);
  }
  if (c==1)
   record_id=1;
  if (c==0)
   goto final;
  j = record_id - 1;
  image_id=get_image_id(j,image_id);
```

```
   for (i=0;i<n;i++) {
     if (strcmp(satt[i].data_type,"image")==0) {
strcpy(table_array[table_index].table_name, satt[i].t_name);
found = check_table_name();
table_cursor = table_entry;
strcpy(media_name,satt[i].a_name);
get_media_name();
display_photo(i,j,temp_table,image_id);
     }
   }
   printf("\nDo you want to see more image data ? (Y/N) :");
   Ans=yes_no_answer();
   if ((Ans==121)||(Ans==89))
     print_result_table(temp_table,flag);
   if ((Ans==110)||(Ans==78))
     goto next;
  }
}
 }
 next:
  for (k=0;k<n;k++) {
    if (strcmp(satt[k].data_type,"sound")==0) {

  Ans =121;
  while ((Ans == 121) || (Ans == 89)){
    print_result_table(temp_table, flag);
    if (c>1){
      printf("\nWhich tuple's sound do you want to hear? (enter record id) :");
      scanf("%d", &record_id);
    }
    if (c==1){
      record_id=1;
    }
    if (c==0)
      goto final;
    j = record_id - 1;
    sound_id=get_sound_id(j,sound_id);
    for (i=0;i<n;i++) {
      if (strcmp(satt[i].data_type,"sound")==0) {
printf("\nSound management");
strcpy(table_array[table_index].table_name, satt[i].t_name);
found = check_table_name();
table_cursor = table_entry;
strcpy(media_name,satt[i].a_name);
get_media_name();
display_sound(i,j,temp_table, sound_id);
```

```
    }
   }
   printf("\nDo you want to hear more sound data ? (Y/N) :");
   Ans=yes_no_answer();
  }
 }
  }
}/* end if ans=121 (the one at  he top) */
else
 k=900;
}/*end if strcmp(datatype=image or sound) */
}/* end for k<n (top one ) */
 /*printf("\n");*/
/* Drop table result after finished print  */
/*drop_table(temp_table);*/
 final:
drop_temp_media_tables();
}
/*******************************************
This function drops a table in the INGRES catalog mgmt usually refers to the temp tables.
*******************************************/
drop_table(table_name)
char table_name[20];
{
 {
   IIsqInit((char *)0);
   IIwritedb("destroy ");
   IIwritedb(table_name);
   IIsqSync(0,(char *)0);
 }
}
/*********************
This function initializes an array upto size 100.
*********************/
init_buffer(buffer,j)
char buffer[100];
int j;
{
 int i;
 for (i=0;i<j;i++) {
   buffer[i] = '\0';
 }
}
/********************
This function drops the temporary media tables used to hold the intermediate results of a query.
To be processed by INGRES, had to call inttostr -integer to string function
```

```
*********************/
drop_temp_media_tables()
{
 int k;
 char l[5];
 char tempstring[100];
 for (k=0; k<10; k++){
  strcpy(tempstring, "p");
  inttostr(k,l);
  strcat(tempstring,l);
  IIsqInit((char *)0);
  IIwritedb("destroy ");
  IIwritedb(tempstring);
  IIsqSync(0,(char *)0);
  init_buffer(tempstring,100);
  init_buffer(l,5);
 }
}

/**************************************
This function asks the user to enter a join condition for retrieval conditions with AND in them.
***************************************/
void help_join()
{
 int i=0;
 if (m > 1) {
  strcpy(join_condition,"?");
  while (strcmp(join_condition,"?")==0) {
   printf("\nPlease enter your join condition\n(<?> for help!) :");
   gets(join_condition);
   if (strcmp(join_condition,"?")==0) {
for (i=0;i<m;i++) {
 printf("\nTable %s ",stab[i].t_name);
 p_att(stab[i].t_name);
} /* end for loop */
   } /* end if need help for join */
  }/*end while*/
 } /* end if more than 1 table select */
}
/************************************
This function asks the user to enter three temp table names for intersection.Two for the different
conditions to be intersected and the third for the result table of the intersection.
*************************************/
char get_temp_table_names_for_intersection(temp_table1,temp_table2,temp_table)
char temp_table1[20];
char temp_table2[20];
```

```
char temp_table[20];
{
 printf("\nEnter first temp table name :");
 gets(buff);
 strcpy(temp_table1, buff);
 init_buffer(buff,100);
 printf("\nEnter second temp table name :");
 gets(buff);
 strcpy(temp_table2, buff);
 init_buffer(buff,100);
 printf("\nEnter another temporary table name to hold the result :");
 gets(buff);
 strcpy(temp_table, buff);
 init_buffer(buff,100);

 return(temp_table1, temp_table2, temp_table);

}
/************************************
This function shows the user choice intersect/union/minus menu.
************************************/
char intersect_union_menu(answer)
char answer;
{
 answer = '?';
/*  while (!( '0'<= answer && answer <= '3'))
  {*/
    clr_scr();
printf("\nIf you want to intersect / union / minus any two temporary tables:\n");
    printf("\t==================\n");
    printf("\n\t1. INTERSECT two tables");
    printf("\n\t2. UNION two tables");
    printf("\n\t3. MINUS");
    printf("\n\t0. Quit");
    printf("\n\t==================\n");
    printf("\n\tSelect your choice :: ");
    answer = getchar();
    while ((c = getchar()) != '\n')
    ; /* Not return do nothing */
/*  } */
  return (answer);

}
/*****************************************************
This function asks the user if he wants to union/intersect/minus any two tables and puts the result
in temp_table.All the function calls for the operations are executed here.
```

```
*****************************************************/
query_for_intersect_union(choice,temp_table1,temp_table2,temp_table)
char choice;
char temp_table1[20];
char temp_table2[20];
char temp_table[20];
{
  choice = '?';
  clr_scr();
  while (choice != '0')
  {
    choice = intersect_union_menu(choice);/* print the choice for user select on screen */
    switch(choice)/* User select case */
    {
 case '1' :/* create table */
  clr_scr();
            printf("\nYour Selection is INTERSECT");
            printf("\nHit Return to continue! (Any other key to QUIT!)");
            if (getchar() != '\n')
              {
               getchar();  /* To let next getchar() work well */
               break;
              }
  get_temp_table_names_for_intersection(temp_table1, temp_table2, temp_table);
  printf("\n*** The result of the INTERSECTION will be kept in temp_table*** %s ***\n",
temp_table);
  intersect_tables(temp_table1, temp_table2, temp_table);
  ql_printdata(temp_table);
  break;
 case '2' :
  clr_scr();
            printf("\nYour Selection is UNION");
            printf("\nHit Return to continue! (Any other key to QUIT!)");
            if (getchar() != '\n')
              {
               getchar();  /* To let next getchar() work well */
               break;
              }
  get_temp_table_names_for_intersection(temp_table1, temp_table2, temp_table);
  printf("\n*** The result of the UNION will be kept in temp_table*** %s ***\n", temp_table);
  union_tables_for_demo(temp_table1, temp_table2, temp_table);
  ql_printdata(temp_table);
  break;
 case '3' :/* create table */
  clr_scr();
            printf("\nYour Selection is MINUS");
```

```
              printf("\nHit Return to continue! (Any other key to QUIT!)");
              if (getchar() != '\n')
                {
                 getchar();  /* To let next getchar() work well */
                 break;
                }
       get_temp_table_names_for_intersection(temp_table1, temp_table2, temp_table);
       printf("\n*** The result of the MINUS will be kept in temp_table*** %s ***\n", temp_table);
       minus(temp_table1, temp_table2, temp_table);
       ql_printdata(temp_table);
       break;

      case '0' :
       clr_scr();
              printf("\nHit Return to continue! (Any other key to QUIT!)");
              if (getchar() != '\n')
                {
                 getchar();  /* To let next getchar() work well */
                 break;
                }
       break;
     } /* End of switch */
   } /* End of while choice != '0' */
   return(choice);
   return(temp_table1);
   return(temp_table2);
   return(temp_table);
}
/************************************
This function displays the user choice different  Retrieval operations menu
************************************/
char show_utility_menu(answer)
char answer;
{
  answer = '?';
/*  while (!( '0'<= answer && answer <= '4'))
   {*/
    clr_scr();
    printf("\n\tRetrieval Operations Menu\n");
    printf("\t==========================\n");
    printf("\n\t0. Simple Condition");
    printf("\n\t1. table1 where EXISTS table2");
    printf("\n\t2. table1 where NOT EXISTS table2");
    printf("\n\t3  table1 IN table2");
    printf("\n\t4. table1 NOT IN table2");
    printf("\n\t=========================\n");
```

```
    printf("\n\tSelect your choice :: ");
    answer = getchar();
    while ((c = getchar()) != '\n')
    ; /* Not return do nothing */
/*  } */
  return (answer);

}
/******************************************************
Ths function calls the function show_utility_menu and calls other functions to process the user's
choice.
******************************************************/
utility_menu(choice,temp_table1,temp_table2,temp_table)
char choice;
char temp_table1[20];
char temp_table2[20];
char temp_table[20];
{
  choice = '?';
  clr_scr();

/*  while (choice != '0')
  {*/
    choice = show_utility_menu(choice);/* print the choice for user select on screen */
    switch(choice)/* User select case */
    {
 case '1' :/* create table */
 clr_scr();
            printf("\nYour Selection is table1 where EXISTS table2");
            printf("\nHit Return to continue! (Any other key to QUIT!)");
            if (getchar() != '\n')
              {
               getchar(); /* To let next getchar() work well */
               break;
              }
 printf("\nEnter the temp table name related to EXISTS :");
 gets(buff);
 strcpy(temp_table2, buff);
 init_buffer(buff,100);
 printf("\nPlease enter your join condition\nbetween ");
 if (m==1)
   printf("%s and ", temp_table1);
 if (m>1)
   printf("the appropriate table and ");
 printf("** %s ** :", temp_table2);
 gets(buff);
```

```
 strcpy(join_for_nested, buff);
 init_buffer(buff,100);
 break;
case '2' :
 clr_scr();
          printf("\nYour Selection is table1 where NOT EXISTS table2");
          printf("\nHit Return to continue! (Any other key to QUIT!)");
          if (getchar() != '\n')
           {
            getchar();  /* To let next getchar() work well */
            break;
           }
 printf("\nEnter the temp table name related to NOT EXISTS :");
 gets(buff);
 strcpy(temp_table2, buff);
 init_buffer(buff,100);
 printf("\nPlease enter your join condition\nbetween ");
 if (m==1)
 printf("%s and ", temp_table1);
 if (m>1)
  printf("the appropriate table and ");
 printf("** %s ** :", temp_table2);
 gets(buff);
 strcpy(join_for_nested, buff);
 init_buffer(buff,100);
 break;
case '3' :
 clr_scr();
          printf("\nYour Selection is table1 IN table2");
          printf("\nHit Return to continue! (Any other key to QUIT!)");
          if (getchar() != '\n')
           {
            getchar();  /* To let next getchar() work well */
            break;
           }
 printf("\nEnter the temp table name related to IN :");
 gets(buff);
 strcpy(temp_table2, buff);
 init_buffer(buff,100);

 printf("\n\nEnter attribute for ");
 if (m==1)
 printf("table %s", temp_table1);
 if (m>1)
  printf("the appropriate table");
 printf(" for condition of IN :");
```

```
 gets(buff);
 strcpy(condition_for_nested, buff);
 init_buffer(buff,100);

 printf("\n\nTable ** %s **", temp_table2);
 printf("\nSELECT ATTRIBUTE (only one attribute!) :");
 gets(buff);
 strcpy(attribute_for_nested, buff);
 init_buffer(buff,100);
 break;
case '4' :
 clr_scr();
           printf("\nYour Selection is table1 NOT IN table2");
           printf("\nHit Return to continue! (Any other key to QUIT!)");
           if (getchar() != '\n')
             {
              getchar(); /* To let next getchar() work well */
              break;
             }
 printf("\nEnter the temp table name related to NOT IN :");
 gets(buff);
 strcpy(temp_table2, buff);
 init_buffer(buff,100);

 printf("\n\nEnter attribute for ");
 if (m==1)
 printf("table %s", temp_table1);
 if (m>1)
   printf("the appropriate table");
 printf(" for condition of NOT IN :");
 gets(buff);
 strcpy(condition_for_nested, buff);
 init_buffer(buff,100);

 printf("\n\nTable ** %s **", temp_table2);
 printf("\nSELECT ATTRIBUTE (only one attribute!) :");
 gets(buff);
 strcpy(attribute_for_nested, buff);
 init_buffer(buff,100);
 break;
case '0' :
 clr_scr();
           printf("\nYour Selection is NORMAL RETRIEVAL");
           printf("\nHit Return to continue! (Any other key to QUIT!)");
           if (getchar() != '\n')
             {
```

```
                getchar(); /* To let next getchar() work well */
                break;
              }
   break;
      } /* End of switch */
   /* }*/ /* End of while choice != '0' */

   return(choice);
   return(temp_table1);
   return(temp_table2);
   return(temp_table);
}
/**********************************
This function checks if any attributes with aggregate functions exist in the attributes entered by
the user.
**********************************/
char check_aggregate(buffer, tmp, aggregate_found)
char buffer[13];
char tmp[3];
{
  int i = 0;
  int jj = 0;
  for (jj=0;jj<3;jj++){
    if (buffer[i]==40){
/*      tmp[jj]='\0';*/
      jj=1000;
    }
    else{
      tmp[jj]=buffer[i];
    }
    i++;
  } /* end for jj < 3 */
  tmp[3]='\0';
  if ((strcmp(tmp,"cnt")==0)||(strcmp(tmp,"sum")==0)||(strc-
mp(tmp,"avg")==0)||(strcmp(tmp,"min")==0)||(strcmp(tmp,"max")==0)){
    aggregate_found=TRUE;
  }
  return(aggregate_found);
}
/**************************************
When there is an aggregate function among the attributes entered by the user, this function sepa-
rates the attribute from the aggregate part.
**************************************/
char get_attribute(buffer, attribute)
char buffer[13];
char attribute[13];
```

```
{
 int i = 4;
 int j;
 for (j=0;j<13;j++){
  if (buffer[i]==41){
   attribute[j]= '\0';
   j=100;
  }
  else{
   attribute[j]=buffer[i];
  }
  i=i+1;
 } /* end for j < 13 */
 return(attribute);
}
/*****************************************
When mod is modify mode (MOD_MODE) this function is the main function calling other other
functions to delete the tuples from the related media tables.
*****************************************/
void delete_for_modify(r)
int r;
{
   int j=0,k=0,l=0,temp;
   char char_value[21],a;
   char file_name[20];
   int  integer_value,media_value,found,media1_value;
   int im_value, so_value;
   int desired_tuple;
   float real_value;
   int i=0,select=0;
   int c=0;
   desired_tuple=r;   /* keeps track of number of tuples */
   printf("\nTuple # %d is being deleted now ...", desired_tuple+1);
   sleep(2);
   /* # line 3169 "db.sc" */      /* select */
   {
       IIsqInit((char *)0);
  IIwritedb("retrieve unique(c=(count("); /* counts the number of tuples in the temp table */
       IIwritedb(temp_table);
       IIwritedb(".");
       IIwritedb(satt[0].a_name);
       IIwritedb(")))");
       IIsqRinit((char *)0);
       if (IIerrtest() == 0) {
        if (IInextget() != 0) {
          IIretdom(1,30,4,&c);
```

```
      } /* IInextget */
          IIsqFlush((char *)0);
    } /* IIerrtest */
   }
    l=0;
    if (c==0) {
            printf("\nPress ENTER to continue...");
            a=getchar();
            return;
    } /*********/
   /* # line 3171 "db.sc" */    /* host code  retrieves the user conditions data */
      if (IIcsrOpen((char *)0,"cursor_output","db1",0,temp_table) != 0) {
IIwritedb("retrieve (");
for (select=0;select<n-1;select++) {
 IIwritedb(satt[select].a_name);
 IIwritedb("=");
 IIwritedb(temp_table);
 IIwritedb(".");
 IIwritedb(satt[select].a_name);
 IIwritedb(",");
}
IIwritedb(satt[select].a_name);
IIwritedb("=");
IIwritedb(temp_table);
IIwritedb(".");
IIwritedb(satt[select].a_name);
IIwritedb(")");
IIcsrQuery((char *)0);
      } /* IIcsrOpen */
    printf("\n");
    look_more=0;
    l=0;
    if (c==0) {
      look_more=1;
    }
   /* Fetch the cursor to the temp_tablerelation which is the intermediate table
        hold the temp_tablefrom the query, then print out the tuple one at a time   until no more
record to print to the user */
    while (look_more == 0) {
      if (IIcsrFetch((char *)0,"cursor_output","db1") != 0) {
if (desired_tuple == 1){
 printf("record id %d \t",l+1);
}
for (i=0;i<n;i++) {
    if (strcmp(satt[i].data_type,"c20")==0) {
      IIcsrRet(1,32,0,char_value);
```

```
    if (desired_tuple == l)
printf("%s :  %s",satt[i].a_name,char_value);
   }
   if (strcmp(satt[i].data_type,"integer")==0) {
    IIcsrRet(1,30,4,&integer_value);
    if (desired_tuple == l)
printf("%s :  %d  ",satt[i].a_name,integer_value);
   }
   if (strcmp(satt[i].data_type,"float")==0) {
    IIcsrRet(1,31,4,&real_value);
    if (desired_tuple == l)
printf("%s :  %8.2f  ",satt[i].a_name,real_value);
   }
   if (strcmp(satt[i].data_type,"image")==0) {
    IIcsrRet(1,30,4,&media_value);
    if (desired_tuple == l){
im_value=media_value;
printf("%s id is %d  ",satt[i].a_name,media_value);
     }
   }
   if (strcmp(satt[i].data_type,"sound")==0) {
    IIcsrRet(1,30,4,&media1_value);
    if (desired_tuple == l){
so_value=media1_value;
printf("%s %d",satt[i].a_name,media1_value);
     }
   }
  } /* end for select < n*/
  printf("\n");
   IIcsrEFetch((char *)0);  /* fetch the next record to the cursor */
l++;  /* increment l as the counter */
if (l==c) {   /* check if no more data to print */
 look_more =1;  /* exit of the loop */
}
     } /* IIcsrFetch */
    } /* end while */
   IIcsrClose((char *)0,"cursor_output","db1"); /* close the cursor */
   printf("Press ENTER to continue ..");
   a= getchar();
   /* this for the check for the media selection */
   if (c==0)
     i=9999; /* if no record for the media data not process any thing */
   for (i=0;i<n;i++) {
     if (strcmp(satt[i].data_type,"image")==0) {
if (image_flag==TRUE){
 strcpy(table_array[table_index].table_name, satt[i].t_name);
```

```
 found = check_table_name(); /* search for the media name */
 table_cursor = table_entry;
 strcpy(media_name,satt[i].a_name);
 get_media_name();
 printf("\nThe media data from the media table *** %s *** is being deleted now...", media_n-
ame);
 sleep(4);
 mod_get_rid_image(i, im_value);
}
    }
    if (strcmp(satt[i].data_type,"sound")==0) {
if (sound_flag==TRUE){
 strcpy(table_array[table_index].table_name, satt[i].t_name);
       found = check_table_name();
 table_cursor = table_entry;
 strcpy(media_name,satt[i].a_name);
 get_media_name();
 printf("\nThe media data from the media table *** %s *** is being deleted now...", media_n-
ame);
 sleep(4);
 mod_get_rid_sound(i, so_value);
}
    }
   } /* end for select < n*/
     printf("\n");
 }
/**********************************************
When mode is MODIFY, this function gets sound file attributes from the related media table.
**********************************************/
get_snd_file_atts(media_name, i, value)
STR_name media_name;
int i;
int value;
{
 int entry;
 char sound_value[20];


 int att_cursor;
 int desired_tupleno;
 char query_phrase[DESCRLEN+1],
    in_phrase[DESCRLEN+1];
 int j=0, k, c, pid, query_err, query_len, in_len, f_flag,look_more=0;

 char ISfn5[FILENAMELEN+1];
 char ISdescr1[DESCRLEN+1];
```

```
int ISerror;
STR_path file_name;
STR_descrp nothing;
char temp_file[100]; /* Declare more to avoid bus error */
int show_pid, wait_pid;
union wait status;
int sid = 0;
int pp=0;
int qq=0;
int res=0;
int sz=0;
int s_rate=0;
int enc;
int dur=0;


inttostr(value, sound_value);
{
  IIsqInit ((char *)0);
  IIwritedb("retrieve unique(pp=(count(");
  IIwritedb(temp_table);
  IIwritedb(".");
  IIwritedb(satt[i].a_name);
  IIwritedb(")))");
  IIsqRinit((char *)0);
  if (IIerrtest()==0) {
    if (IInextget() !=0) {
IIretdom(1,30,4,&pp);
    }
    IIsqFlush((char *)0);
  }
}

{
  if (IIcsrOpen((char *)0,"cursor_output8","db3",0,media_name) != 0) {
    IIwritedb("retrieve(ISfn5=");
    IIwritedb(media_name);
    IIwritedb(".");
    IIwritedb("f_id,ISdescr1=");
    IIwritedb(media_name);
    IIwritedb(".descrp,");

    IIwritedb("res=");
    IIwritedb(media_name);
    IIwritedb(".");
    IIwritedb("resolution,");
```

```
        IIwritedb("sz=");
        IIwritedb(media_name);
        IIwritedb(".");
        IIwritedb("size,");

        IIwritedb("s_rate=");
        IIwritedb(media_name);
        IIwritedb(".");
        IIwritedb("samp_rate,");

        IIwritedb("enc=");
        IIwritedb(media_name);
        IIwritedb(".");
        IIwritedb("encoding, ");

        IIwritedb("sid=");
        IIwritedb(media_name);
        IIwritedb(".");
        IIwritedb("s_id,");

        IIwritedb("dur=");
        IIwritedb(media_name);
        IIwritedb(".");
        IIwritedb("duration");

        IIwritedb(")");
        IIwritedb(" where ");
        IIwritedb(media_name);
        IIwritedb(".s_id=");
        IIwritedb(sound_value);
        IIcsrQuery ((char *)0);
      }
    }
    pp=1;
    {
        while (look_more==0) {
  if (IIcsrFetch((char *)0, "cursor_output8","db3") != 0) {

    IIcsrRet(1,32,0,ISfn5);
    IIcsrRet(1,32,0,ISdescr1);
    IIcsrRet(1,30,4,&res);
    IIcsrRet(1,30,4,&sz);
    IIcsrRet(1,30,4,&s_rate);
    IIcsrRet(1,30.4,&enc);
```

```
 IIcsrRet(1,30,4,&sid);
 IIcsrRet(1,30,4,&dur);

 strcpy(file_name, ISfn5);
 strcpy(snd_record[snd_index].f_id, file_name);
 strcpy(descrp, ISdescr1);
 strcpy(snd_record[snd_index].descrp, descrp);
 snd_record[snd_index].resolution = res;
 snd_record[snd_index].size = sz;
 snd_record[snd_index].samp_rate = s_rate;
 snd_record[snd_index].encoding = enc;
 snd_record[snd_index].s_id = sid;
 snd_record[snd_index].duration = dur;

 snd_value[snd_index]=snd_record[snd_index].s_id;/*-------*/
 att_array[att_cursor].value_entry=snd_index;

 printf("\n");
 IIcsrEFetch((char *)0);
 qq++;
 if (qq==pp) {
   look_more = 1;
 }
}
    }
    IIcsrClose((char *)0,"cursor_output8","db3");
   }

 init_buffer(sound_value, 20);
}
/*********************************************
When mode is MODIFY, this function gets the image file atts from the related media table.
*********************************************/
get_file_id(media_name, i, value)
STR_name media_name;
int i;
int value;
{
 int entry;
 int att_cursor;
 int desired_tupleno;
 int k=0, j=0, look_more=0;
 char ISfn1[FILENAMELEN+1];
 char ISdescr1[DESCRLEN+1];

 char image_value[20];
```

```
int hght = 0;
int wdth = 0;
int dpth = 0;
int iid = 0;

STR_path f_name;
STR_descrp nothing;
char temp_file[100]; /* Declare more to avoid bus error */
struct pixrect *pr;
colormap_t cm;
int show_pid, wait_pid;
union wait status;
int over_length = TRUE; /* Initialize to true */
cm.type = RMT_NONE;     /* this is absolutely necessary! Otherwise     */
cm.length = 0;          /* pr_load_colormap might not allocate storage */
cm.map[0] = NULL;       /* for the colormap, if the garbage found in   */
cm.map[1] = NULL;       /* the cm structure seems to make sense. The   */
cm.map[2] = NULL;       /* result, of course, is segmentation fault.   */


inttostr(value, image_value);
{
 IIsqInit ((char *)0);
 IIwritedb("retrieve unique(k=(count(");
 IIwritedb(media_name);
 IIwritedb(".");
 IIwritedb("i_id");
 IIwritedb(")))");
 IIsqRinit((char *)0);
 if (IIerrtest()==0) {
  if (IInextget() !=0) {
IIretdom(1,30,4,&k);
  }
  IIsqFlush((char *)0);
 }
}

{
 if (IIcsrOpen((char *)0,"cursor_output1","db",0,media_name) != 0) {
  IIwritedb("retrieve(ISfn1=");
  IIwritedb(media_name);
  IIwritedb(".");
  IIwritedb("f_id,ISdescr1=");
  IIwritedb(media_name);
```

```
    IIwritedb(".descrp,");

    IIwritedb("hght=");
    IIwritedb(media_name);
    IIwritedb(".");
    IIwritedb("height,");

    IIwritedb("iid=");
    IIwritedb(media_name);
    IIwritedb(".");
    IIwritedb("i_id,");

    IIwritedb("wdth=");
    IIwritedb(media_name);
    IIwritedb(".");
    IIwritedb("width,");

    IIwritedb("dpth=");
    IIwritedb(media_name);
    IIwritedb(".");
    IIwritedb("depth");

    IIwritedb(")");
    IIwritedb(" where ");
    IIwritedb(media_name);
    IIwritedb(".");
    IIwritedb("i_id");
    IIwritedb("=");
    IIwritedb(image_value);
    IIcsrQuery ((char *)0);
   }
  }
  k=1;/*--------------------------------*/
  {
    while (look_more==0) {
if (IIcsrFetch((char *)0, "cursor_output1","db") != 0) {
  IIcsrRet(1,32,0,ISfn1);
  IIcsrRet(1,32,0,ISdescr1);

  IIcsrRet(1,30,4,&hght);
  IIcsrRet(1,30,4,&iid);
  IIcsrRet(1,30,4,&wdth);
  IIcsrRet(1,30,4,&dpth);


  strcpy(f_name, ISfn1);
```

```
 strcpy(img_record[img_index].f_id, f_name);
 strcpy(descrp, ISdescr1);
 strcpy(img_record[img_index].descrp, descrp);
 img_record[img_index].height = hght;
 img_record[img_index].i_id = iid;
 img_record[img_index].width = wdth;
 img_record[img_index].depth = dpth;
 /*----------------------------------*/
 img_value[img_index]=img_record[img_index].i_id;
 att_array[att_cursor].value_entry=img_index;
 /*----------------------------------*/
 printf("\n");
 IIcsrEFetch((char *)0);
 j++;
 if (j==k) {
  look_more = 1;
 }
}
    }
    IIcsrClose((char *)0,"cursor_output1","db");
   }
/* printf("\nimg_record[img_index].i_id =>%d",img_record[img_index].i_id);
 printf("\nimg_record[img_index].f_id =>%s",img_record[img_index].f_id);
 printf("\nimg_record[img_index].descrp =>%s",img_record[img_index].descrp);
 sleep(1);*/
 init_buffer(image_value, 20);
}
/*********************************************
When mode is modify, this function helps user modify the tuples in the result table one by one.
*********************************************/
process_tuple_by_tuple(r)
int r;
{
 int entry;
 int desired_tuple;
 char c_temp[60];
 int count=0;
 int j=0,k=0,l=0,temp;
 char char_value[21],a;
 char file_name[20];
 int img_value, snd_value:
 int  integer_value,media_value,found,media1_value;
 float real_value;
  int i=0,select=0;
 int g=0;
 int d;
```

```
 i_value[i_index]=0;
 desired_tuple=r;
 printf("\nTuple to be modified :: Tuple # %d ", desired_tuple+1);
 sleep(3);
/* # line 3169 "db.sc" *//* select */
 {
  IIsqInit((char *)0);
  IIwritedb("retrieve(g=(count(");
  IIwritedb(temp_table);
  IIwritedb(".");
  IIwritedb(satt[0].a_name);
  IIwritedb(")))");
  IIsqRinit((char *)0);
  if (IIerrtest() == 0) {
    if (IInextget() != 0) {
      IIretdom(1,30,4,&g);
    } /* IInextget */
    IIsqFlush((char *)0);
  } /* IIerrtest */
 }
l=0;
if (g==0) {
printf("\nPress ENTER to continue...");
a=getchar();
return;
}
/* # line 3171 "db.sc" *//* host code */
  if (IIcsrOpen((char *)0,"cursor_output","db2",0,temp_table) != 0) {
    IIwritedb("retrieve(");
    for (select=0;select<n-1;select++) {
    IIwritedb(satt[select].a_name);
    IIwritedb("=");
    IIwritedb(temp_table);
    IIwritedb(".");
    IIwritedb(satt[select].a_name);
    IIwritedb(",");
}
    IIwritedb(satt[select].a_name);
    IIwritedb("=");
    IIwritedb(temp_table);
    IIwritedb(".");
    IIwritedb(satt[select].a_name);
    IIwritedb(")");
    IIcsrQuery((char *)0);
  } /* IIcsrOpen */
 printf("\n");
```

```
look_more=0;
l=0;
if (g==0) {
 look_more=1;
}
table_cursor = table_entry;
count=0;
count = table_array[table_list[table_cursor]].att_count;
att_cursor = table_array[table_list[table_cursor]].att_entry;
act_media_count = 0;
i_index=0;
c_index=0;
/* Fetch the cursor to the result relation which is the intermediate table
   hold the result from the query, then print out the tuple one at a time
   until no more record to print to the user */

 while (look_more == 0) {
  if (IIcsrFetch((char *)0,"cursor_output","db2") != 0) {
   if (desired_tuple == 1){
printf("record id %d \t",l+1);
    }
for (i=0;i<n;i++) {
 if (strcmp(satt[i].data_type,"c20")==0) {
  IIcsrRet(1,32,0,char_value);
  if (desired_tuple == 1){
   printf("%s :  %s",satt[i].a_name,char_value);
   strcpy(c_temp, char_value);
   strcpy(c_value[c_index], c_temp);
   att_array[att_cursor].value_entry = c_index;
   c_index = (c_index + 1) % 20;
   att_cursor = att_array[att_cursor].next_index;
  }
 }
 if (strcmp(satt[i].data_type,"integer")==0) {
  IIcsrRet(1,30,4,&integer_value);
  if (desired_tuple == 1){
   printf("%s :  %d  ",satt[i].a_name,integer_value);
   i_value[i_index]=integer_value;
   att_array[att_cursor].value_entry = i_index;
   i_index = (i_index + 1) % 20;
   att_cursor = att_array[att_cursor].next_index;
  }
 }
 if (strcmp(satt[i].data_type,"float")==0) {
  IIcsrRet(1,31,4,&real_value);
  if (desired_tuple == 1){
```

```
      printf("%s :  %8.2f  ",satt[i].a_name,real_value);
     }
    }
    if (strcmp(satt[i].data_type,"image")==0) {
     IIcsrRet(1,30,4,&media_value);
     if (desired_tuple == l){
      img_value=media_value;
      printf("%s id is %d  ",satt[i].a_name,media_value);
     }
    }
    if (strcmp(satt[i].data_type,"sound")==0) {
     IIcsrRet(1,30,4,&media1_value);
     if (desired_tuple == l){
      snd_value=media1_value;
      printf("%s %d",satt[i].a_name,media1_value);
     }
    }
   }/*end for select n*/
      printf("\n");
      IIcsrEFetch((char *)0);  /* fetch the next record to the cursor */
      l++;  /* increment l as the counter */
      if (l==g) {   /* check if no more data to print */
   look_more =1;  /* exit of the loop */
       }
     } /* IIcsrFetch */
    } /* end while */

    IIcsrClose((char *)0,"cursor_output","db2"); /* close the cursor */

    printf("Press ENTER to continue ..");
    a= getchar();

    for (i=0;i<n;i++) {
     if (strcmp(satt[i].data_type,"image")==0) {
      strcpy(table_array[table_index].table_name, satt[i].t_name);
      found = check_table_name(); /* search for the media name */
      table_cursor = table_entry;
      strcpy(media_name,satt[i].a_name);
      get_media_name();
      printf("\nThe attribute values will be read from the media table  %s", media_name);
      sleep(2);
      get_file_id(media_name, i, img_value);
      printf("Press ENTER to continue ..");
      a= getchar();
      att_cursor = att_array[att_cursor].next_index;
      media_counter++;
```

```
   media_value=0;/*--------------*/
  }
  if (strcmp(satt[i].data_type,"sound")==0) {
   strcpy(table_array[table_index].table_name, satt[i].t_name);
   found = check_table_name();
   table_cursor = table_entry;
   strcpy(media_name,satt[i].a_name);
   get_media_name();
   printf("\nThe attribute values will be read from the media table  %s", media_name);
   sleep(2);
   get_snd_file_atts(media_name, i, snd_value);
   printf("Press ENTER to continue ..");
   a= getchar();
   att_cursor = att_array[att_cursor].next_index;
   media_counter++;
   media1_value=0;/*--------------*/
  }
 } /* end for select < n*/

}
/*****************************************************************
When mode is MODIFY, this function prints the number of tuples in the result table.
*****************************************************************/
int print_for_modify(c)
int c;
{
 c=0;
 /* # line 3169 "db.sc" *//* select */
 {
  IIsqInit((char *)0);
  IIwritedb("retrieve(c=(count(");
  IIwritedb(temp_table);
  IIwritedb(".");
  IIwritedb(satt[0].a_name);
  IIwritedb(")))");
  IIsqRinit((char *)0);
  if (IIerrtest() == 0) {
   if (IInextget() != 0) {
    IIretdom(1,30,4,&c);
   } /* IInextget */
   IIsqFlush((char *)0);
  } /* IIerrtest */
 }
 printf("\n*** THERE ARE %d RECORDS (TUPLES) TO BE MODIFIED ***",c);
 printf("\n(You will be queried for modifying each tuple)");
 sleep(3);
```

```
 return(c);
}
/********************
When mode is modify, this function deletes the modified tuples from the tables.
********************/
void delete_formatted_part_for_modify()
{
 int i;
 printf("\nThe tuples that match the delete query are being deleted from table *** %s *** now",-
satt[0].t_name);
 printf("\nPress ENTER to continue");
 a=getchar();
 IIsqInit((char *)0);
 IIwritedb("delete ");
 IIwritedb(satt[0].t_name);
 IIwritedb(" where ");
 for (i=0; i<n-1; i++){
  IIwritedb(satt[0].t_name);
  IIwritedb(".");
  IIwritedb(satt[i].a_name);
  IIwritedb("=");
  IIwritedb(temp_table);
  IIwritedb(".");
  IIwritedb(satt[i].a_name);
  IIwritedb(" and ");
 }
  IIwritedb(satt[0].t_name);
  IIwritedb(".");
  IIwritedb(satt[i].a_name);
  IIwritedb("=");
  IIwritedb(temp_table);
  IIwritedb(".");
  IIwritedb(satt[i].a_name);

 IIwritedb(" ");
 IIsqSync(1,(char *)0);
}


/*******************************************************************
This function, when mode is DELETE, deletes the  from the related media tables.
*******************************************************************/
get_rid_image(imageno)
int imageno;
{
 IIsqInit((char *)0);
```

```
 IIwritedb("delete ");
 IIwritedb(media_name);
 IIwritedb(" where ");
 IIwritedb(media_name);
 IIwritedb(".");
 IIwritedb("i_id =");
 IIwritedb(temp_table);
 IIwritedb(".");
 IIwritedb(satt[imageno].a_name);
 IIwritedb(" ");
 IIsqSync(1,(char *)0);
}
/*********************************************************************
This function, when mode is MODIFY, deletes the modified tuples from the related media tables.
*********************************************************************/
mod_get_rid_image(imageno, value)
int imageno;
int value;

{
 char media_value[20];
 inttostr(value, media_value);
 {
  IIsqInit((char *)0);
  IIwritedb("delete ");
  IIwritedb(media_name);
  IIwritedb(" where ");
  IIwritedb(media_name);
  IIwritedb(".");
  IIwritedb("i_id");
  IIwritedb("=");
  IIwritedb(media_value);
  IIwritedb(" ");
  IIsqSync(1,(char *)0);
 }
}
/*********************************************************************
This function, when mode is DELETE, deletes the tuples from the related media tables.
*********************************************************************/
get_rid_sound(soundno)
int soundno;
{

 {
  IIsqInit((char *)0);
  IIwritedb("delete ");
```

```
   IIwritedb(media_name);
   IIwritedb(" where ");
   IIwritedb(media_name);
   IIwritedb(".");
   IIwritedb("s_id =");
   IIwritedb(temp_table);
   IIwritedb(".");
   IIwritedb(satt[soundno].a_name);
   IIwritedb(" ");
   IIsqSync(1,(char *)0);

  }
}
/***********************************
This function, when mode is MODIFY, deletes the modified tuples from the related media tables.
***********************************/
mod_get_rid_sound(soundno, value)
int soundno;
int value;
{
 char sound_value[20];
 inttostr(value, sound_value);
  {
   IIsqInit((char *)0);
   IIwritedb("delete ");
   IIwritedb(media_name);
   IIwritedb(" where ");
   IIwritedb(media_name);
   IIwritedb(".");
   IIwritedb("s_id");
   IIwritedb("=");
   IIwritedb(sound_value);
   IIwritedb(" ");
   IIsqSync(1,(char *)0);
  }
}
/***************************************
When mode is DELETE, this functions is the main function calling other functions to delete the
tuples from the related media tables.
***************************************/
void ql_print_delete_data()
{
   int j=0,k=0,l=0,temp;
     char char_value[21],a;
     char file_name[20];
     int  integer_value,media_value,found,media1_value;
```

```
   float real_value;
    int i=0,select=0;
  int c=0;
  /* # line 3169 "db.sc" */      /* select */
  {
       IIsqInit((char *)0);
 IIwritedb("retrieve unique(c=(count(");
       IIwritedb(temp_table);
       IIwritedb(".");
       IIwritedb(satt[0].a_name);
       IIwritedb(")))");
       IIsqRinit((char *)0);
       if (IIerrtest() == 0) {
         if (IInextget() != 0) {
           IIretdom(1,30,4,&c);
     } /* IInextget */
         IIsqFlush((char *)0);
   } /* IIerrtest */
  }
   l=0;
  printf("\nThere are %d records that match the DELETE query",c);
  if (c==0) {
        printf("\nPress ENTER to continue...");
        a=getchar();
        return;
   }  /*********/
 /* # line 3171 "db.sc" */    /* host code */
    if (IIcsrOpen((char *)0,"cursor_output","db1",0,temp_table) != 0) {
IIwritedb("retrieve (");
for (select=0;select<n-1;select++) {
 IIwritedb(satt[select].a_name);
 IIwritedb("=");
 IIwritedb(temp_table);
 IIwritedb(".");
 IIwritedb(satt[select].a_name);
 IIwritedb(",");
}
IIwritedb(satt[select].a_name);
IIwritedb("=");
IIwritedb(temp_table);
IIwritedb(".");
IIwritedb(satt[select].a_name);
IIwritedb(")");
IIcsrQuery((char *)0);
    } /* IIcsrOpen */
  printf("\n");
```

```
  look_more=0;
  l=0;
  if (c==0) {
   look_more=1;
  }
 /* Fetch the cursor to the temp_tablerelation which is the intermediate table
     hold the temp_tablefrom the query, then print out the tuple one at a time   until no more
record to print to the user */
 while (look_more == 0) {
     if (IIcsrFetch((char *)0,"cursor_output","db1") != 0) {
       printf("record id %d\t",l+1);
      for (i=0;i<n;i++) {
if (strcmp(satt[i].data_type,"c20")==0) {
 IIcsrRet(1,32,0,char_value);
 printf("%s :  %s",satt[i].a_name,char_value);
 }
if (strcmp(satt[i].data_type,"integer")==0) {
 IIcsrRet(1,30,4,&integer_value);
 printf("%s :  %d  ",satt[i].a_name,integer_value);
 }
if (strcmp(satt[i].data_type,"float")==0) {
 IIcsrRet(1,31,4,&real_value);
 printf("%s :  %8.2f  ",satt[i].a_name,real_value);
      }
if (strcmp(satt[i].data_type,"image")==0) {
 IIcsrRet(1,30,4,&media_value);
 printf("%s id is %d  ",satt[i].a_name,media_value);
 }
if (strcmp(satt[i].data_type,"sound")==0) {
 IIcsrRet(1,30,4,&media1_value);
 printf("%s %d",satt[i].a_name,media1_value);
      }
     } /* end for select < n*/
    printf("\n");
    IIcsrEFetch((char *)0);  /* fetch the next record to the cursor */
    l++;  /* increment l as the counter */
     if (l==c) {   /* check if no more data to print */
       look_more =1;  /* exit of the loop */
      }
   } /* IIcsrFetch */
 } /* end while */
 IIcsrClose((char *)0,"cursor_output","db1"); /* close the cursor */
     printf("Press ENTER to continue ..");
     /* stop before change to the next function so
        the user can see the temp_tableon screen, until he hit ENTER key */
     a= getchar();
```

```
 /* this for the check for the media selection */
 if (c==0)
  i=9999; /* if no record for the media data not process any thing */
  for (i=0;i<n;i++) {
   if (strcmp(satt[i].data_type,"image")==0) {
if (image_flag==TRUE){
 strcpy(table_array[table_index].table_name, satt[i].t_name);
 found = check_table_name(); /* search for the media name */
 table_cursor = table_entry;
 strcpy(media_name,satt[i].a_name);
 get_media_name();
 printf("\nmedia_name--> ***%s***", media_name);
 sleep(2);
 get_rid_image(i);
}
   }
   if (strcmp(satt[i].data_type,"sound")==0) {
if (sound_flag==TRUE){
 strcpy(table_array[table_index].table_name, satt[i].t_name);
     found = check_table_name();
 table_cursor = table_entry;
 strcpy(media_name,satt[i].a_name);
 get_media_name();
 printf("\nmedia_name--> ***%s***", media_name);
 sleep(2);
 get_rid_sound(i);
}
   }
  } /* end for select < n*/
    printf("\n");
 }
/*******************************************************************
When mode is MODIFY, this function checks the media description if the media data is modifed.
*******************************************************************/
int mod_chk_description(file_id, descrp, err_message)
STR_path *file_id;
STR_descrp *descrp;
char *err_message;
{
 int i=0;
 int error = FALSE;
 while (i<1 && !error){
  *err_message = '\0';
  if (strcmp(descrp, " ") != 0)
   error = connect_parser(file_id, descrp, err_message);
  i++;
```

```
 }

 if (error)
  {
   printf("\nThe description for media is NOT acceptable!");

   if (error == DESCR_WORD_ERR)
    printf("\nThe system cannot understand the word >>%s<<", err_message);
   else
    if (error == DESCR_STRUCTURE_ERR)
     printf("\nThe system cannot interpret the phase\n >>%s<<",
err_message);
    else
     printf("\nThe program error occurred in prolog!\n");
   printf("\nPlease modify it. Thank you!");
   putchar('\007');
   while((c=getchar()) != '\n')
   ;
   return(TRUE);
  }
 else
  return(FALSE);
}
/*********************************
Gets all atts of a given table and puts them in satt array for retrieving all the attributes of that
table.Used in delete and modify procedures.
*********************************/
void get_all_atts_of_a_given_table()
{
 int i = 0,
   count = 0;
 count = table_array[table_list[table_cursor]].att_count;
 n = count;
 att_cursor = table_array[table_list[table_cursor]].att_entry;
 for (i = 0; i < count; i++) /*  Loop to get value for each attribute */
  {
   strcpy(satt[i].t_name, stab[0].t_name);
   strcpy(satt[i].a_name, att_array[att_cursor].att_name);
   strcpy(satt[i].data_type, att_array[att_cursor].data_type);
   att_cursor = att_array[att_cursor].next_index;
  } /* End of for loop */
} /* End of get_tuple_value */

/
***************************************************************************
/
```

```
/* The main procedure for the retrieve operation                    */
/* m and n is the parameter for table and attribute repectively      */
/* For retrieve table name and attribute name from the user          */
/*This function also handles DELETE and some of the MODIFY operations           */
/*******************************************************************************/
void retrieve(mode)
{
 int entry;
 int count;
 int h,r,flag=TRUE;
 int o, u;
 char buf0[13];
 char buf1[13];
 char buf2[13];
 char buf3[13];
 char buf4[13];
 char temp[3];
 char aggregate0[3];
 char aggregate1[3];
 char aggregate2[3];
 char aggregate3[3];
 char aggregate4[3];
 int i,j,x,y,z,found=0;
 int level_no=0, counter=1;
 char table_name[20],attname[20],att_type[20],Ans,More,a;
 char choice;
 init_buffer(buff,100);
 init_buffer(temp_table1,20);
 init_buffer(temp_table2,20);
 init_buffer(temp_table,20);
 choice='0';
 m=0;
 i=0;
 k=0;
 gcond=0;
 numcon=0;
 aggregate_found=FALSE;
 more_selections=TRUE;
 more_levels=TRUE;

 init();
 drop_temp_media_tables();

 while (more_levels != FALSE){
  while (more_selections != FALSE){
   init_buffer(buff,100);
```

```
    printf("\nEnter table name to hold the temporary result of the query: ");
    gets(buff);
    strcpy(temp_table, buff);
    init_buffer(buff,100);

    help_tables(buff);
    while (i<=table_count) {/* check loop with the maximum number table */
for (j=0;j<13;j++)/* each table has less than or equal to 12 char only */
 {
   if (buff[k]==44) {
    stab[i].t_name[j]= '\0';
    j=55;
    k=k+1;
    i=i+1;
   }
   else {
    if (buff[k] == ' ')
j=55;  /* Skip the white space if the user typped in*/
    else
stab[i].t_name[j]=buff[k];
    if (buff[k]==0) { /* if null value in buffer (end of string) */
m=i+1;
j=55;
i=1000;
    }

    k=k+1;
    }
 }
    }/*end while*/

    strcpy(temp_table1, stab[0].t_name);

    for (i=0;i<m;i++) {
strcpy(table_array[table_index].table_name, stab[i].t_name);
found = check_table_name(); /* search for the media name */
if (!(found)) {
/* check for the valid table name if not found then return to calling program */
  putchar('\007');
  printf("\nTable %s not found please redo again !!!" ,stab[i].t_name);
  printf("\nPress ENTER to continue !!");
  a=getchar();
  return;
} /* end else */
    } /* end for loop */
```

```
/* Specify the join condition if there are more than 2 table select */
    help_join();

/* Select attribute */
    init_buffer(buff,100);
    i = 0;
    j = 0;
    k = 0;
    x = 0;
    z = 0;
    if (mode == RTRVE_MODE){
/* Select attribute for one table at a time */
for (y=0;y<m;y++) {
  printf("\nTable %s ", stab[y].t_name);
  strcpy(buff,"?");
  while (strcmp(buff,"?")==0) {
   printf("\nSelect the attribute(s) separated by comma <,> - <?> for HELP ! -\nhit <ESC> for no
attribute");
   printf("\nSELECT ATTRIBUTE(S) : ");
   gets(buff);
   if (strcmp(buff,"?")==0) {
    p_att(stab[y].t_name);
   } /* end if buff == "?"  */
  } /* end while need help */

  while (i < 100) {
   for (j=0;j<13;j++){
    if (buff[k]==27) {
goto start_again;
    }
    if (buff[k]==44) {
buf0[j]= '\0';
strcpy(satt[x].t_name, stab[y].t_name);
init_buffer(temp,3);
init_buffer(aggregate0,3);
u=x;
aggregate_found=FALSE;
aggregate_found=check_aggregate(buf0, temp, aggregate_found);
printf("\n");
strcpy(aggregate0, temp);
if (aggregate_found==TRUE){
  get_attribute(buf0, satt[u].a_name);
  printf("\n");
  if (strcmp(aggregate0,"cnt")==0)
    satt[u].aggregate_type=1;
  if (strcmp(aggregate0,"sum")==0)
```

```
  satt[u].aggregate_type=2;
 if (strcmp(aggregate0,"avg")==0)
  satt[u].aggregate_type=3;
 if (strcmp(aggregate0,"max")==0)
  satt[u].aggregate_type=4;
 if (strcmp(aggregate0,"min")==0)
  satt[u].aggregate_type=5;
 printf("\n");
}
if (aggregate_found==FALSE){
 strcpy(satt[u].a_name,buf0);
 satt[u].aggregate_type=0;
 printf("\n");
 clr_scr();
}
j=55;
k=k+1;
i=i+1;
x=x+1;
    }
    else {
if (buff[k] == ' ')
 j=55; /* Skip the white space if user typped in */
else{
 buf0[j]=buff[k];
}
if (buff[k]==0) {
 strcpy(satt[x].t_name, stab[y].t_name);
 init_buffer(temp,3);
 init_buffer(aggregate0,3);
 u=x;
 aggregate_found=FALSE;
 aggregate_found=check_aggregate(buf0, temp, aggregate_found);
 printf("\n");
 strcpy(aggregate0, temp);
 if (aggregate_found==TRUE){
  get_attribute(buf0, satt[u].a_name);
  printf("\n");
  if (strcmp(aggregate0,"cnt")==0)
   satt[u].aggregate_type=1;
  if (strcmp(aggregate0,"sum")==0)
   satt[u].aggregate_type=2;
  if (strcmp(aggregate0,"avg")==0)
   satt[u].aggregate_type=3;
  if (strcmp(aggregate0,"max")==0)
   satt[u].aggregate_type=4;
```

```
  if (strcmp(aggregate0,"min")==0)
    satt[u].aggregate_type=5;
  printf("\n");
 }
 if (aggregate_found==FALSE){
  strcpy(satt[u].a_name,buf0);
  satt[u].aggregate_type=0;
  printf("\n");
  clr_scr();
 }

 n=x+1;
 j=55;
 i=1000;
 }
k=k+1;
    } /* end else */
   } /* end for j < 13 */
 }/*end while */
 x=x+1;
start_again:
 k=0;
 init_buffer(buff,100);
 i=0;
} /* End select attribute for each table go to the next table */

clr_scr();
for (i=0;i<n;i++) {
 printf("\n%s.%s", satt[i].t_name,satt[i].a_name);
 getatttype(satt[i].t_name,satt[i].a_name,satt[i].data_type);
}
    } /*  closure of if mod ==ret  */
    if ((mode==DEL_MODE) || (mode==MOD_MODE)){
table_cursor = table_entry;
get_all_atts_of_a_given_table();
    }

    printf("\n");
    cond=0;
    printf("\nAny condition ? (y/n) :");
    Ans=yes_no_answer();
    if ((Ans==121)||(Ans==89)){
choice=nested_processcondition(choice,temp_table1,temp_table2,temp_table);
    }

    if (choice=='0'){
```

```
ql_retrieve(temp_table);
ql_printdata(temp_table);
    }
    init_buffer(buff,100);

    query_for_intersect_union(choice,temp_table1,temp_table2,temp_table);
    printf("\nMore selections at this level ? (y/n)");
    Ans=yes_no_answer();
    if ((Ans==121)||(Ans==89)){
more_selections=TRUE;
choice='0';
y = 0;
j = 0;
x = 0;
z = 0;
m=0;
i=0;
k=0;
cond=0;
gcond=0;
numcon=0;
n=0;
found=0;
init();
drop_temp_media_tables();
init_buffer(buff,100);
init_buffer(temp_table1,20);
init_buffer(temp_table2,20);
init_buffer(temp_table,20);
    }
    else{
more_selections=FALSE;
printf("\n");
    }
    }
    printf("\nMore levels ? (y/n)");
Ans=yes_no_answer();
if ((Ans==121)||(Ans==89)){
  more_levels=TRUE;
  more_selections=TRUE;
  level_no=level_no+1;
  choice='0';
  y = 0;
  j = 0;
  x = 0;
  z = 0;
```

```
  m=0;
  i=0;
  k=0;
  cond=0;
  gcond=0;
  numcon=0;
  n=0;
  init();
  drop_temp_media_tables();
  init_buffer(buff,100);
  init_buffer(temp_table1,20);
  init_buffer(temp_table2,20);
  init_buffer(temp_table,20);
  found=0;
}
else{
  more_selections=FALSE;
  more_levels=FALSE;
}
    }/* end while more levels */

    if (mode==DEL_MODE){
image_flag=TRUE;
sound_flag=TRUE;
printf("\nDo want to continue with DELETION ? (y/n) ::");
Ans=yes_no_answer();
if ((Ans==110)||(Ans==78))
  goto qquit;
ql_print_delete_data();
delete_formatted_part_for_modify();
drop_table(temp_table);
image_flag = FALSE;
sound_flag = FALSE;
    }
    if (mode==MOD_MODE){
formatted_flag = FALSE;
image_flag = FALSE;
sound_flag = FALSE;
h=print_for_modify(h);
for (r=0; r<h; r++){
  formatted_flag = FALSE;
  image_flag = FALSE;
  sound_flag = FALSE;
  media_counter = 0;
  process_tuple_by_tuple(r);
  mod_display_tuple(mode, media_counter);
```

```
 store_data(mode);
 mod_ql_insert_tuple(mode);
 att_cursor = 0;/*to initialize the value arrays */
 img_index = 0;
 snd_index = 0;
 i_index = 0;
 f_index = 0;
 c_index = 0;
 delete_for_modify(r);
}
delete_formatted_part_for_modify();
drop_table(temp_table);
image_flag = FALSE;
sound_flag = FALSE;
    }
   qquit:
    printf("\n ");
} /* End procedure */
```

APPENDIX B: SOURCE CODE FOR THE MODIFICATION

The programming code for the modify operation is contained in this appendix.

```
/************************** ModifyModule.c *********************************
* Title        : ModifyModule.c                                            *
* Author       : Aygun/Stewart                                             *
* Date         : July 1991                                                 *
* History      : Created based on the PEI insert module modifcations       *
*              : prior to the insertion into the MDBMS and the             *
*              : Retrieval operations developed by Pongsuwan and Aygun     *
*              :                                                           *
* Description  : This module implements the modification of tuples        *
*              : existing (already stored in the catalog management        *
*              :MDBMS.                                                     *
***************************************************************************/
*Export Interface :                                                        *
*  mod_print_all_table():Prints out the table catalog information on screen *
*  mod_insert_tuple()  :Inserts a tuple of a particular relation           *
*     display_tuple()   :Displays the tuple before insertion               *
* check_media_descrp():Checks media description by connecting to the parser *
*      ql_insert_tuple():Translates SQL statement to insert a standart tuple *
*      show_image()   :Displays image on the screen by passing pixels and colormap from *
*                       the caller. It might be able to quit the image automatically before *
*                       displaying the next image.                         *
****************************************************************************
* Import Interface:                                 *
*   mod_check_table_name():Checks table name to see if it is duplicate   *
*      get_media_name() :Gets media table name by appending table_key at *
*                 the end of att_name.              *
*                 from CreateModule.c                 *
*                                               *
*   mod_get_sound_value() :Gets a sound value of a media attribute from the *
*                 user input                      *
*      yes_no_answer()  :Gets yes or no answer from the user        *
*      clr_scr()        :Clears the screen.               *
*                 from UserInterface.c               *
***************************************************************************/
#include <stdio.h>
#include <string.h>
#include <pixrect/pixrect_hs.h>
#include <sys/wait.h>
#include <suntool/sunview.h>
```

```
#include <suntool/canvas.h>

#include "defines.h"
#include "errors.h"
#include "struct.h"
#include "GlobalVariables.h"


char c;
struct SND_HDR s_hdr;
int image_counter=141;
int snd_ctr = 142;
int modify_mode;
int formatted_flag = FALSE;
int image_flag = FALSE;
int sound_flag = FALSE;
int act_media_count;
int act_media_list[10];
/*************************************************************************/
/* Print out the table catalog information on screen for the modification    */
/* operations                                                             */
/*************************************************************************/
void mod_print_all_table()
{
  int i = 0;
  printf("\t**Table Name**\n");
  for (i = 0; i < table_count; i++)
   {
    printf("\t  %s\n",table_array[table_list[i]].table_name);
    if ((i % 15) == 14)
{
 printf("\n*RETURN TO CONTINUE*\n");
 while ((c = getchar()) != '\n')
 ;
 printf("\t**Table Name**\n");
}
   } /* End of for loop */
 } /* End of print_all_table() */

/*************************************************************************/
/* Get a INTEGER value of a standard attribute from the user input for the */
/* modification operations
/*************************************************************************/
void mod_get_int_value()
 {
  char stuff[3]; /* To provide a dummy var for '\n' when user enter '?' */
```

```
 i_value[i_index] = 0;
 scanf("%d", &i_value[i_index]);
 if (i_value[i_index] == 0)        /* ? or 0 entered */
  {
   i_value[i_index] = 0;         /* if 0 entered still 0 */
   stuff[0] = '\0';
   gets(stuff); /* To let next gets() work when ? entered in scanf() */
  }
 else
  getchar(); /* Add after scanf() to let next gets() work properly */
 att_array[att_cursor].value_entry = i_index;
 i_index = (i_index + 1) % 20;
} /* End of mod_get_int_value() */

/***************************************************************************/
/* Get a FLOAT value of a standard attribute from the user input          */
/***************************************************************************/
void mod_get_float_value()
{
 char stuff[3]; /* To provide a dummy var for '\n' when user enter '?' */
 f_value[f_index] = 0.0;
 scanf("%f", &f_value[f_index]);
 if (f_value[f_index] == 0.0)      /* ? or 0 entered */
  {
   f_value[f_index] = 0.0;       /* if 0 entered still 0.0 */
   stuff[0] = '\0';
   gets(stuff); /* To let next gets() work when ? entered in scanf() */
  }
 else
  getchar(); /* Add after scanf() to let next gets() work properly */
 att_array[att_cursor].value_entry = f_index;
 f_index = (f_index + 1) % 20;
} /* End of mod_get_float_value() */
/***************************************************************************/
/* Get a STRING value of a standard attribute from the user input         */
/***************************************************************************/
void mod_get_c20_value()
{
  int over_length = TRUE; /* Initialize to true */
  char c_temp[60]; /* Temp var for read in, 60 to avoid bus error */
  while (over_length)
   {
c_temp[0] = '\0';
gets(c_temp);
if (strlen(c_temp) >= 21)
 {
```

```
    printf("\nSorry!! Value OVER 20 characters!");
    putchar('\007');
    printf("\nPlease Enter <<%s>> Value ( ? if unknow):: ",
    data_type);
  }
else
  {
   over_length = FALSE;
   strcpy(c_value[c_index], c_temp);
   if (strcmp(c_value[c_index], "?") == 0)
     strcpy(c_value[c_index], " "); /* Assign blank as null */
   att_array[att_cursor].value_entry = c_index;
   c_index = (c_index + 1) % 20;
  } /* End of if else */
    } /* End of while (over_length) */
} /* End of mod_get_c20_value() */
/**************************************************************************/
/* Get the description of a MEDIA attribute from the user input. Asks the */
/* user for a sentence or a phrase to describe the sound or image data type*/
/* If the input is too long or if there is an empty string an error message*/
/* occurs.                                           */
/**************************************************************************/
void mod_get_descrp()
{
  char phrase[MAX_PHRASE+20]; /* Maximum length of a phrase is 127 */
  int phrase_len = 0,         /* Declared 20 char more to avoid the*/
      descrp_len = 0;         /* bus error!                       */
  int stop_input = FALSE;
  descrp[0] = '\0';
  printf("\nPlease enter your query description\n\
 * Noun phrases separated by commas and ending with an exclamation mark\n\
 * Sentences end with a period.\n\
(end entire description with an empty line):\n");
  while (!stop_input)
   {
    phrase[0] = '\0';
    gets(phrase);
    phrase_len = strlen(phrase);
    if (phrase_len >= 1)
      {
       if (phrase_len >= MAX_PHRASE) /*Need end with \n & \0 in one phrase*/
  {
  printf("\nThe phrase OVER %d characters!", (MAX_PHRASE - 1));
  printf("\nInvalid input!! TRY AGAIN!\n");
  putchar('\007');
  }
```

```
else
 {
  phrase[phrase_len] = '\n';
  phrase[phrase_len + 1] = '\0';
  if (phrase_len > 1)
   {
    if ((descrp_len + phrase_len + 1) >= (MAX_DESCRP + 1))
     {
      stop_input = TRUE;
      printf("\nThe last phrase extended beyond the maximum %d ",
MAX_DESCRP);
      printf("\ncharacters in description. It has been canceled!\n");
      putchar('\007');
      while ((c = getchar()) != '\n')
      ;
     }
    else
     {
      strcat(descrp, phrase);
      descrp_len = descrp_len + phrase_len + 1;
     } /* End of if else */
   }; /* End of if (phrase_len > 1) */
 } /* End of if else (phrase_len >= MAX_PHRASE) */
} /* End of if (phrase_len >= 1) */
      else /* Empty string input */
{
 if (descrp_len == 0)
  {
  printf("\nSorry! Empty string is NOT allowed!\n");
  putchar('\007');
  }
 else
  stop_input = TRUE;
} /* End of if else */
   } /* End of while (!stop_input) */

} /* End of mod_get_descrp() */
/***************************************************************************/
/* Get a SOUND value of a media attribute from the user input to modify    */
/* a sound value already stored in the database. Some error checking.     */
/***************************************************************************/
void mod_get_sound_value()
{
  STR_path file_name;
  char temp_file[100]; /* Declare more to avoid bus error */
  int size = 0,
```

```
    samp_rate = 0,
    encoding = 0,
    resolution = 0;
  float duration = 0.0;
  int over_length = TRUE;  /* Initialize to true */
  snd_record[snd_index].s_id = att_array[att_cursor].media_id;
  while (over_length)
   {
    printf("\nPlease Enter <<%s>> File Name!!", data_type);
    printf("\nNOTE: Enter The Full Path Name:: ( ? if unknow)\n");
    temp_file[0] = '\0';
    gets(temp_file);
    if (strlen(temp_file) >= (MAX_PATH +1))
     {
 printf("\nSorry!! PATH_NAME OVER %d characters! TRY AGAIN!!\n",
MAX_PATH);
 putchar('\007');
     }
    else
     {
 strcpy(file_name, temp_file);
 if (strcmp(file_name, "?") == 0)
  {
   over_length = FALSE;
   strcpy(snd_record[snd_index].f_id, " ");
   strcpy(snd_record[snd_index].descrp, " ");
   snd_record[snd_index].size = size;
   snd_record[snd_index].samp_rate = samp_rate;
   snd_record[snd_index].encoding = encoding;
   snd_record[snd_index].duration = duration;
   snd_record[snd_index].resolution = resolution;
  }
 else
  {
   if ((snd_file = fopen(file_name, "r")) == NULL)
    {
     printf("\n%s", file_name);
     printf("\nThe File cannot be opened! Try Again!!\n");
     putchar('\007');
    }
   else
    {
          s_hdr.sfname[0] = '\0';
     snd_load(file_name);    /*Get registra from sound text file*/
     if (strlen(s_hdr.sfname) != 12)  /* sfname must 12 chars as */
 {                        /* a test of sound file    */
```

```
  printf("\n%s", file_name);
  printf("\nThe File does not contain a proper sound!");
  printf(" Try Again!!\n");
  putchar('\007');
 }
    else                    /* i.e. Valid input */
 {
  over_length = FALSE;
            strcpy(snd_record[snd_index].f_id, s_hdr.sfname);
  printf("\nPlay the sound before enter the description?");
  printf(" (y/n)::");
  if (yes_no_answer() == 'y')
   play_snd();
            snd_record[snd_index].size = s_hdr.s_size;
            snd_record[snd_index].samp_rate = s_hdr.s_samplrate;
            snd_record[snd_index].encoding = s_hdr.s_encoding;
            snd_record[snd_index].duration = s_hdr.s_duration;
            snd_record[snd_index].resolution = s_hdr.s_resolution;
 } /* End of if else */
    } /* End of if else */
   fclose(snd_file);
  } /* End of if else */
     } /* End of if else */
   } /* End of while (over_length) */
 } /* End of mod_get_sound_value() */

/*****************************************************************************/
/* Get a IMAGE value of a media attribute from the user input for modifying*/
/* an existing image value. Some error checking is done on the input.     */
/*****************************************************************************/
void mod_get_image_value()
{
  STR_path file_name;
  STR_descrp nothing;
  char temp_file[100];  /* Declare more to avoid bus error */
  int height = 0,
     width = 0,
     depth = 0;
  struct pixrect *pr;
  colormap_t cm;
  int show_pid, wait_pid;
  union wait status;
  int over_length = TRUE;  /* Initialize to true */
  cm.type = RMT_NONE;      /* this is absolutely necessary! Otherwise     */
  cm.length = 0;           /* pr_load_colormap might not allocate storage */
  cm.map[0] = NULL;        /* for the colormap, if the garbage found in   */
```

```
  cm.map[1] = NULL;       /* the cm structure seems to make sense. The  */
  cm.map[2] = NULL;       /* result, of course, is segmentation fault.  */
  img_record[img_index].i_id = att_array[att_cursor].media_id;
  while (over_length)
   {
    printf("\nPlease Enter <<%s>> File Name!!", data_type);
    printf("\nNOTE: Enter The Full Path Name:: ( ? if unknow)\n");
    temp_file[0] = '\0';
    gets(temp_file);
    if (strlen(temp_file) >= (MAX_PATH +1))
     {
printf("\nSorry!! PATH_NAME OVER %d characters! TRY AGAIN!!\n",
MAX_PATH);
putchar('\007');
     }
    else
     {
strcpy(file_name, temp_file);
if (strcmp(file_name, "?") == 0)
 {
  over_length = FALSE;
  strcpy(img_record[img_index].f_id, " ");
  strcpy(img_record[img_index].descrp, " ");
  img_record[img_index].height = height;
  img_record[img_index].width = width;
  img_record[img_index].depth = depth;
 }
else
 {
  if ((img_file=fopen(file_name, "r")) == NULL)
   {
    printf("\n%s", file_name);
    printf("\nThe File cannot be opened! Try Again!!\n");
    putchar('\007');
   }
  else {
    pr = pr_load(img_file, &cm);  /* Get registration data */
         ISimage_from_pixrect(pr, &cm, file_name, nothing);
    if (pr == NULL)
{
 printf("\n%s", file_name);
 printf("\nThe File does not contain a proper image!");
 printf("\nThe image must be in Sun Raster format!");
 printf(" Try Again!!\n");
 putchar('\007');
}
```

```
    else {
  over_length = FALSE;
  strcpy(img_record[img_index].f_id, file_name);
  printf("\nDisplay the image before enter the description?");
  printf(" (y/n):: ");
  if (yes_no_answer() == 'y')
    show_image(pr, &cm);
  img_record[img_index].height = pr->pr_size.y;
  img_record[img_index].width = pr->pr_size.x;
  img_record[img_index].depth = pr->pr_depth;
} /* End of if else */
    } /* End of if else */
   fclose(img_file);
  } /* End of if else */
     } /* End of if else */
   } /* End of while (over_length) */
} /* End of mod_get_image_value() */
/**************************************************************************/
/* Get a value of a standard attribute from the user input to modify      */
/* existing standard value(s) in the database.                            */
/**************************************************************************/
void mod_get_std_value()
{
  printf("\nTable Name:: %s\nAtt Name  :: %s\nData Type :: %s",
  table_array[table_list[table_cursor]].table_name,
    att_array[att_cursor].att_name,
    att_array[att_cursor].data_type);
  printf("\nPlease Enter <<%s>> Value ( ? if unknow):: ", data_type);
  if (strcmp(data_type, "integer") == 0)
    mod_get_int_value();                /* Integer data type */
  else
    if (strcmp(data_type, "float") == 0)
      mod_get_float_value();            /* Float data tupe */
    else
      mod_get_c20_value();              /* String c20 data tupe */
} /* End of get_std_value() */
/**************************************************************************/
/* Get a value of a media attribute from the user input to modify the data */
/* already stored. Can re-enter the description or filename of either image*/
/* or sound media attributes.                                             */
/**************************************************************************/
void mod_get_media_value()
{
  printf("\nTable Name:: %s\nAtt Name  :: %s\nData Type :: %s",
  table_array[table_list[table_cursor]].table_name,
    att_array[att_cursor].att_name,
```

```
    att_array[att_cursor].data_type);
  if (strcmp(data_type, "image") == 0)
    {
img_value[img_index] = att_array[att_cursor].media_id;
att_array[att_cursor].value_entry = img_index;
mod_get_image_value();                    /* Image data type */
      if (strcmp(img_record[img_index].f_id, " ") != 0)
        {
          printf("\nEnter the description? (y/n):: ");
          if (yes_no_answer() == 'y')
            mod_get_descrp();
          else
            strcpy(descrp, " ");
          strcpy(img_record[img_index].descrp, descrp);
        }
att_array[att_cursor].media_id++;
img_index = (img_index + 1) % 20;
    }
  else
    {
snd_value[snd_index] = att_array[att_cursor].media_id;
att_array[att_cursor].value_entry = snd_index;
mod_get_sound_value();                    /* Sound data tupe */
      if (strcmp(snd_record[snd_index].f_id, " ") != 0)
        {
          printf("\nEnter the description? (y/n):: ");
          if (yes_no_answer() == 'y')
            mod_get_descrp();
          else
            strcpy(descrp, " ");
          strcpy(snd_record[snd_index].descrp, descrp);
        }
att_array[att_cursor].media_id++;
snd_index = (snd_index + 1) % 20;
    } /* End of if else */
} /* End of get_media_value() */
/****************************************************************************/
/* Get the values of a tuple from the user input. It begins to loop at the */
/* 1st attribute until the last attribute entered for that modified tuple */
/****************************************************************************/
void mod_get_tuple_value()
{
  int i = 0,
     count = 0;
  count = table_array[table_list[table_cursor]].att_count;
  att_cursor = table_array[table_list[table_cursor]].att_entry;
```

```
  act_media_count = 0;
  for (i = 0; i < count; i++)  /* Loop to get value for each attribute */
   {
strcpy(data_type, att_array[att_cursor].data_type);
if ((strcmp(data_type, "image") == 0) ||
    (strcmp(data_type, "sound") == 0))
 {
  get_media_value();
  act_media_list[act_media_count] = att_cursor;  /* Collect the  */
  act_media_count++;                              /* media indices*/
 }
else
 get_std_value();
att_cursor = att_array[att_cursor].next_index;
   }  /* End of for loop */
}  /* End of get_tuple_value */
/**************************************************************************/
/* Insert a modified tuple of one particular relation.                    */
/**************************************************************************/
void mod_insert_tuple()
{
  int table_found = FALSE;  /* Initialize to false */
  while (!table_found)
   {
    printf("\nEnter table_name::(Maximum 12 characters); ( ? for HELP!)\n");
    table_name[0] = '\0';
    gets(table_name);
    if (strlen(table_name) >= 13)  /* Over maximum name length */
     {
  printf("\nSorry!! Table Name OVER 12 characters!");
  putchar('\007');
     }
    else
     {
  if (strcmp(table_name, "?") == 0)
   print_all_table();
  else
   {
     strcpy(table_array[table_index].table_name, table_name);
     table_found = check_table_name();
     if (table_found)
          {
   table_cursor = table_entry;
   get_tuple_value();
  }
     else
```

```
 {
  printf("\nSorry!! Table name: %s NOT found! TRY AGAIN!!",
table_array[table_index].table_name);
  putchar('\007');
 } /* End of if else */
    } /* End of if else */
      } /* End of if else */
   } /* End of while (!table_found) */
} /* End of insert_tuple() */
/**************************************************************************/
/* Print out the value of the modified current tuple which the user wants */
/* to insert.                                                             */
/**************************************************************************/
void mod_print_tuple()
{
  int i = 0,
     count = 0,
     entry = 0;
  clr_scr();
  entry = table_array[table_list[table_cursor]].att_entry;
  count = table_array[table_list[table_cursor]].att_count;
  printf("\nTable Name:: %s\n",
     table_array[table_list[table_cursor]].table_name);
  printf("\nNumber  Attribute Name\tData Type\tValue\n");
  for (i = 0; i < count; i++)
   {
    strcpy(data_type, att_array[entry].data_type);
    if (strcmp(data_type, "c20") == 0)
     printf(" %d  %13s\t%s\t\t\'%s\'\n",(i+1), att_array[entry].att_name,
 att_array[entry].data_type,
     c_value[att_array[entry].value_entry]);
    else
     if (strcmp(data_type, "integer") == 0)
      printf(" %d  %13s\t%s\t\t%d\n",(i+1) , att_array[entry].att_name,
      att_array[entry].data_type,
     i_value[att_array[entry].value_entry]);
     else
      if (strcmp(data_type, "float") == 0)
printf(" %d  %13s\t%s\t\t%f\n",(i+1) , att_array[entry].att_name,
att_array[entry].data_type,
     f_value[att_array[entry].value_entry]);
      else
if (strcmp(data_type, "image") == 0)
 {
  printf(" %d  %13s\t%s\t\t",(i+1) , att_array[entry].att_name,
     att_array[entry].data_type);
```

```
  if (strcmp(img_record[att_array[entry].value_entry].f_id, " ")
== 0)
    printf("NO VALUE\n");
  else
    printf("HAS VALUE\n");
 }
else
 {
  printf(" %d  %13s\t%s\t\t",(i+1) , att_array[entry].att_name,
      att_array[entry].data_type);
  if (strcmp(snd_record[att_array[entry].value_entry].f_id, " ")
== 0)
    printf("NO VALUE\n");
  else
    printf("HAS VALUE\n");
 }
    entry = att_array[entry].next_index;
   } /* End of for loop i */
 } /* End of mod_print_tuple() */
/******************************************************************************/
/* Print out the description of media attribute in the  modified tuple        */
/******************************************************************************//
 void mod_print_media_tuple()
 {
  int i = 0,
     entry;
  int s = 1;
  int cont = TRUE;
  int ptter = 0;/*----------added---------*/
  STR_name data_type;
  ptter = table_array[table_list[table_cursor]].att_entry;/*----added---*/
  while (cont){
   att_cursor = ptter;
   strcpy(data_type, att_array[att_cursor].data_type);
   if ((strcmp(data_type, "image") == 0) || (strcmp(data_type, "sound") == 0))
     goto close;
   ptter = att_array[ptter].next_index;
  }
 close:
  att_cursor = ptter;
  entry = att_array[att_cursor].value_entry;
  printf("\nMedia Description::\n");
/*  for (i = 0; i < act_media_count; i++)*/
  for (i = 0; i < media_counter; i++)
   {
    printf("\nAtt_name   :: %s", att_array[att_cursor].att_name);
```

```
    strcpy(data_type, att_array[att_cursor].data_type);
/*    entry = att_array[act_media_list[i]].value_entry;*/
    if (strcmp(data_type, "image") == 0)
     {
 printf("\nFile_name  :: \'%s\'", img_record[entry].f_id);
 printf("\nDescription:: \n<<%s>>", img_record[entry].descrp);
     }
    else
     {
 printf("\nFile_name  :: \'%s\'", snd_record[entry].f_id);
 printf("\nDescription:: \n<<%s>>", snd_record[entry].descrp);
     }
    ptter = att_array[ptter].next_index;
    att_cursor = ptter;
    while ((c = getchar()) != '\n')
    ;
   } /* End of for loop */
 } /* End of mod_print_media_tuple() */

/*************************************************************************/
/* Print out the value of current attribute                             */
/*************************************************************************/
void mod_print_value()
{
  int entry;
  entry = att_array[att_cursor].value_entry;
  clr_scr();
  printf("\nTable Name:: %s",
     table_array[table_list[table_cursor]].table_name);
  printf("\nAtt_Name  :: %s", att_array[att_cursor].att_name);
  printf("\nData Type :: %s", att_array[att_cursor].data_type);
  printf("\nValue     :: ");
  if (strcmp(data_type, "c20") == 0)
    printf("\'%s\'\n", c_value[entry]);
  else
   if (strcmp(data_type, "integer") == 0)
    printf("%d\n", i_value[entry]);
   else
    if (strcmp(data_type, "float") == 0)
printf("%f\n", f_value[entry]);
    else
     if (strcmp(data_type, "image") == 0)
      {
 printf("\n\t==>File_name  :: \'%s\'", img_record[entry].f_id);
 printf("\n\t==>Description:: \n<<%s>>\n", img_record[entry].descrp);
      }
```

```
   else
    {
 printf("\n\t==>File_name :: \'%s\'", snd_record[entry].f_id);
 printf("\n\t==>Description:: \n<<%s>>\n", snd_record[entry].descrp);
    }
} /* End of mod_print_value() */
/*************************************************************************/
/* Connect to parser to generate the facts file. We put the modified media*/
/* description in one facts file "imagei_image_facts" at this time, it    */
/* should be separated later on.                                          */
/*************************************************************************/
int mod_connect_parser(file_id, new_descrp, err_message)
STR_path *file_id;
STR_descrp *new_descrp;
char *err_message;
{
  STR_path nothing;
  STR_descrp empty_descrp;
  int ISerror = FALSE;
  empty_descrp[0] = '\0';
  nothing[0] = '\0';
  printf("\nConnect to PARSER, Please Wait....\n");
  ISerror = ISreplace_description("image", "i_image", file_id, empty_descrp,
                    new_descrp, nothing, empty_descrp, err_message);
    /* HERE, ISfunction call, Connect to parser and generate the */
    /* facts file in "imagei_image_facts"                        */
  if (ISerror){
   return(ISerror);
  }
  else
   return(FALSE);
} /* End of connect_parser() */

/*************************************************************************/
/* Change the IMAGE values of current tuple which already exists in the  */
/* database. The user will insert this modified data.                    */
/*************************************************************************/
void mod_change_img_value(mode)
{
 int cursor; /* Previous index of media record array */
 char *err_message;
 int wrong_desc = TRUE;
 cursor = att_array[att_cursor].value_entry;
 img_value[img_index] = att_array[att_cursor].media_id;
 att_array[att_cursor].value_entry = img_index;
 printf("\nChange IMAGE file name? (y/n):: ");
```

```
 if (yes_no_answer() == 'y')
  mod_get_image_value();  /* Image data type */
 else
  {
   img_record[img_index].i_id = att_array[att_cursor].media_id;
   strcpy(img_record[img_index].f_id, img_record[cursor].f_id);
   img_record[img_index].height = img_record[cursor].height;
   img_record[img_index].width = img_record[cursor].width;
   img_record[img_index].depth = img_record[cursor].depth;
  }
 printf("\nChange IMAGE description? (y/n):: ");
 if (yes_no_answer() == 'y')
  {
   mod_get_descrp();
   strcpy(img_record[img_index].descrp, descrp);
   if (mode == MOD_MODE){
printf("\nf_id==>%s", img_record[img_index].f_id);
printf("\ndescrp==>%s", descrp);
 wrong_desc=mod_chk_description(img_record[img_index].f_id, img_record[img_index].de-
scrp, err_message);
while (wrong_desc){
 mod_get_descrp();
 strcpy(img_record[img_index].descrp, descrp);
 wrong_desc=mod_chk_description(img_record[img_index].f_id, img_record[img_index].de-
scrp, err_message);
}
if (!wrong_desc)
 {
  printf("\n\nHit RETURN to Continue!!");
  while ((c = getchar()) != '\n');
 };
   }
  }
 else
  strcpy(img_record[img_index].descrp, img_record[cursor].descrp);
 att_array[att_cursor].media_id++;
 img_index = (img_index + 1) % 20;
} /* End of mod_change_img_value() */

/***************************************************************************/
/* Change the SOUND values of current tuple which the user wants to modify */
/***************************************************************************/
void mod_change_snd_value(mode)
{
 int cursor;  /* Previous index of media record array */
 char *err_message;
```

```
int wrong_desc = TRUE;
cursor = att_array[att_cursor].value_entry;
snd_value[snd_index] = att_array[att_cursor].media_id;
att_array[att_cursor].value_entry = snd_index;
printf("\nChange SOUND file name? (y/n):: ");
if (yes_no_answer() == 'y')
  mod_get_sound_value(); /* Sound data type */
else
  {
   snd_record[snd_index].s_id = att_array[att_cursor].media_id;
   strcpy(snd_record[snd_index].f_id, snd_record[cursor].f_id);
   snd_record[snd_index].size = snd_record[cursor].size;
   snd_record[snd_index].samp_rate = snd_record[cursor].samp_rate;
   snd_record[snd_index].encoding = snd_record[cursor].encoding;
   snd_record[snd_index].duration = snd_record[cursor].duration;
   snd_record[snd_index].resolution = snd_record[cursor].resolution;
  }
printf("\nChange SOUND description? (y/n):: ");
if (yes_no_answer() == 'y')
  {
   mod_get_descrp();
   strcpy(snd_record[snd_index].descrp, descrp);
   if (mode == MOD_MODE){
printf("\nf_id==>%s", snd_record[snd_index].f_id);
printf("\ndescrp==>%s", descrp);
wrong_desc=mod_chk_description(snd_record[snd_index].f_id, snd_record[snd_index].descrp,
err_message);
while (wrong_desc){
 mod_get_descrp();
 strcpy(snd_record[snd_index].descrp, descrp);
 wrong_desc=mod_chk_description(snd_record[snd_index].f_id, snd_record[snd_index].descrp,
err_message);
}
if (!wrong_desc)
 {
  printf("\n\nHit RETURN to Continue!!");
  while ((c = getchar()) != '\n');
 };
   }
  }
 else
  strcpy(snd_record[snd_index].descrp, snd_record[cursor].descrp);
 att_array[att_cursor].media_id++;
 snd_index = (snd_index + 1) % 20;
} /* End of mod_change_snd_value() */
/****************************************************************************/
```

```
/* Change the values of current tuple which the user want to modify      */
/**************************************************************************/
void mod_modify_tuple(mode)
{
  int i = 0,
      count = 0,
      entry = 0,
      order = 0;
  char more_change = 'y';
  while (more_change == 'y')
  {
    mod_print_tuple();
    printf("Select the order which you want to change its value::\n");
    printf("Any other key to cancel the operation!! Select::");
    scanf("%d", &order);
    getchar(); /* To let next gets() work properly */
    entry = table_array[table_list[table_cursor]].att_entry;
    count = table_array[table_list[table_cursor]].att_count;
    if (1 <= order && order <= count)
     {
for (i = 1; i < order; i++)
  entry = att_array[entry].next_index;
att_cursor = entry; /* Assign the current index of att_array */
strcpy(data_type, att_array[att_cursor].data_type);
mod_print_value();
printf("\nPlease Enter <<%s>> Value ( ? if unknow):: ", data_type);
if (strcmp(data_type, "integer") == 0){
  mod_get_int_value(); /* Integer data type */
  formatted_flag=TRUE;
}
else
 if (strcmp(data_type, "float") == 0){
   mod_get_float_value(); /* Float data tupe */
  formatted_flag=TRUE;
 }
 else
  if (strcmp(data_type, "c20") == 0){
    mod_get_c20_value(); /* String c20 data tupe */
  formatted_flag=TRUE;
  }
  else
   if (strcmp(data_type, "image") == 0){
          mod_change_img_value(mode);
    image_flag=TRUE;
/*--------------------------------*/
    act_media_list[act_media_count]=entry;
```

```
    act_media_count++;
/*---------------------------------*/
  }
       else{
         mod_change_snd_value(mode);
    sound_flag=TRUE;
/*---------------------------------*/
    act_media_list[act_media_count]=entry;
    act_media_count++;
/*---------------------------------*/

  }
      mod_print_value();
    }
   else
    {
printf("\nSorry! You entered the wrong order!! Please redo again.\n");
putchar('\007');
    } /* End of if else */
   printf("Any More Change? (y/n):: ");
   more_change = yes_no_answer();
  } /* End of while */
} /* End of mod_modify_tuple() */
/**************************************************************************/
/* Display the modified tuple before insertion                         */
/**************************************************************************/
void mod_display_tuple(mode)
{
  char modify = 'y';
  while (modify == 'y')
  {
    clr_scr();
    mod_print_tuple();
    while ((c= getchar()) != '\n')
    ;
    if (media_counter >= 1)
mod_print_media_tuple();
    printf("\nAny change before insert? (y/n)::");
    modify = yes_no_answer();
    if (modify == 'y')
mod_modify_tuple(mode);
  } /* End of while */
} /* End of display_info() */


/**************************************************************************/
```

```
/* Check the media description by connecting to parser                    */
/*****************************************************************************/
int mod_check_media_descrp()
{
  int i = 0,
      entry;
  int error = FALSE;
  char *err_message;
  while (i < act_media_count && !error)
   {
    *err_message = '\0';
    strcpy(data_type, att_array[act_media_list[i]].data_type);
    entry = att_array[act_media_list[i]].value_entry;
    if (strcmp(data_type, "image") == 0)
      {
       if (strcmp(img_record[entry].descrp, " ") != 0)
         error = connect_parser(img_record[entry].f_id,
                         img_record[entry].descrp, err_message);
      }
    else
      {
       if (strcmp(snd_record[entry].descrp, " ") != 0)
         error = connect_parser(snd_record[entry].f_id,
                         snd_record[entry].descrp, err_message);
      }
    i++;
   }
  if (error)
   {
    printf("\nThe description for media \'%s\' is NOT acceptable!",
                         att_array[act_media_list[i-1]].att_name);
    if (error == DESCR_WORD_ERR)
      printf("\nThe system cannot understand the word >>%s<<", err_message);
    else
      if (error == DESCR_STRUCTURE_ERR)
        printf("\nThe system cannot interpret the phase\n >>%s<<",
 err_message);
      else
        printf("\nThe program error occur in prolog!\n");
    printf("\nPlease modify it. Thank you!");
    putchar('\007');
    while((c=getchar()) != '\n')
    ;
    return(TRUE);
   }
  else
```

```
  return(FALSE);
} /* End of check_media_descrp() */
/****************************************************************************/
/* Translate SQL statement to insert a media tuple                          */
/****************************************************************************/
void mod_ql_insert_media_tuple(mode, image_counter, snd_ctr)
{
 int i = 0,
     entry;
 int enter=0;
/*  printf("\nformatted_flag==>%d", formatted_flag);
  printf("\nimage_flag==>%d", image_flag);
  printf("\nsound_flag==>%d", sound_flag);
  printf("\nact_media_count in ql_ins_media_t==>%d", act_media_count);
  sleep(2);*/
  for (i = 0; i < act_media_count; i++)
   {
    strcpy(media_name, att_array[act_media_list[i]].att_name);
    get_media_name();

    strcpy(data_type, att_array[act_media_list[i]].data_type);
    entry = att_array[act_media_list[i]].value_entry;
    if (strcmp(data_type, "image") == 0){
printf("\n ");
if (image_flag == TRUE){
 printf("  insert into %12s (", media_name);
 printf("i_id   ,\n                  ");
 printf("f_id   ,\n                  ");
 printf("descrp ,\n                  ");
 printf("height ,\n                  ");
 printf("width  ,\n                  ");
 printf("depth  )\n");
 printf("          values(");


 printf("  %d ,\n                 ",
                 image_counter);

 printf("\'%s\',\n                ",
   img_record[entry].f_id);
 printf("\'%s\',\n                ",
   img_record[entry].descrp);
 printf("  %d ,\n                 ",
   img_record[entry].height);
 printf("  %d ,\n                 ",
   img_record[entry].width);
```

```
 printf("  %d );\n\n", img_record[entry].depth);
}
else{
 printf("\nPress ENTER to continue");
 while ((c = getchar()) != '\n')
   ;
}

    }
    if (strcmp(data_type, "sound") == 0){
printf("\n ");
if (sound_flag == TRUE){
 printf("  insert into %12s (", media_name);
 printf("s_id     ,\n                    ");
 printf("f_id     ,\n                    ");
 printf("descrp   ,\n                    ");
 printf("size     ,\n                    ");
 printf("samp_rate,\n                    ");
 printf("encoding ,\n                    ");
 printf("duration ,\n                    ");
 printf("resolution)\n");
 printf("          values (");

 printf("  %d ,\n                   ",
                snd_ctr);

 printf("\'%s\',\n                  ",
   snd_record[entry].f_id);
 printf("\'%s\',\n                  ",
   snd_record[entry].descrp);
 printf("  %d ,\n                   ",
   snd_record[entry].size);
 printf("  %d ,\n                   ",
   snd_record[entry].samp_rate);
 printf("  %d ,\n                   ",
   snd_record[entry].encoding);
 printf("  %f ,\n                   ",
   snd_record[entry].duration);
 printf("  %d );\n\n", snd_record[entry].resolution);
}
else{
 printf("\nPress ENTER to continue");
 while ((c = getchar()) != '\n')
   ;
}
    }
```

```
/**INSERTION OF THE MODIFIED  MEDIA TUPLE ININGRES  STARTS  HERE ***/
/***********THE INGRES FUNCTION CALLS WRITE MANUALLY****************/
/* # line 2100 "db.sc" */   /* insert */
  {

    IIsqInit(&sqlca);
    IIwritedb("append to ");
    IIwritedb(media_name);
    IIwritedb("(");
    if (strcmp(data_type, "image") == 0){
      if (image_flag == TRUE){
printf("\nINSERTING MEDIA TUPLE NOW. PLEASE WAIT!!\n");
IIwritedb("i_id=");
IIsetdom(1,30,4, &image_counter);
IIwritedb(" ,f_id=");
IIsetdom(1,32,0, img_record[entry].f_id);
IIwritedb(" ,descrp=");
IIsetdom(1,32,0, img_record[entry].descrp);
IIwritedb(" ,height=");
IIsetdom(1,30,4, &img_record[entry].height);
IIwritedb(" ,width=");
IIsetdom(1,30,4, &img_record[entry].width);
IIwritedb(" ,depth=");
IIsetdom(1,30,4, &img_record[entry].depth);
IIwritedb(" )");
printf("\nINSERT AN IMAGE TUPLE COMPLETE!!\n");
      }
    }
    if (strcmp(data_type, "sound") == 0){
      if (sound_flag == TRUE){
printf("\nINSERTING MEDIA TUPLE NOW. PLEASE WAIT!!\n");
IIwritedb("s_id=");
IIsetdom(1,30,4, &snd_ctr);
IIwritedb(" ,f_id=");
IIsetdom(1,32,0, snd_record[entry].f_id);
IIwritedb(" ,descrp=");
IIsetdom(1,32,0, snd_record[entry].descrp);
IIwritedb(" ,size=");
IIsetdom(1,30,4, &snd_record[entry].size);
IIwritedb(" ,samp_rate=");
IIsetdom(1,30,4, &snd_record[entry].samp_rate);
IIwritedb(" ,encoding=");
IIsetdom(1,30,4, &snd_record[entry].encoding);
IIwritedb(" ,duration=");
IIsetdom(1,31,4, &snd_record[entry].duration);
IIwritedb(" ,resolution=");
```

```
IIsetdom(1,30,4, &snd_record[entry].resolution);
IIwritedb(" )");
printf("\nINSERT A SOUND TUPLE COMPLETE!!\n");
    }
   }
   IIsqSync(3,&sqlca);
  }
/* # line 2147 "db.sc" */   /* insert */
/***********INSERT MEDIA TUPLE IN INGRES STOP HERE******************/
    printf("\npress ENTER to continue");
    while ((c = getchar()) != '\n')
    ;
   } /* End of for loop */
} /* End of mod_ql_insert_media_tuple() */
/****************************************************************************/
/* Translate SQL statement to insert a modified standard tuple            */
/****************************************************************************/
void mod_ql_insert_tuple(mode)
{
  int i = 0,
     count = 0,
     entry = 0;
  clr_scr();
  entry = table_array[table_list[table_cursor]].att_entry;
  count = table_array[table_list[table_cursor]].att_count;
/*  printf("\nmode in ql_ins_t==>%d", mode);
  printf("\nact_media_count in ql_ins_t==>%d", act_media_count);
  sleep(2);*/
  printf("\nSQL statement::\n");
  printf("  insert into %12s (",
table_array[table_list[table_cursor]].table_name);
  for (i = 1; i < count; i++)
   {
    printf("%12s,\n", att_array[entry].att_name);
    printf("                        ");
    entry = att_array[entry].next_index;
   }
  printf("%12s)\n", att_array[entry].att_name);
  printf("          values (");
  entry = table_array[table_list[table_cursor]].att_entry;
  for (i = 1; i < count; i++)
   {
    strcpy(data_type, att_array[entry].data_type);
    if (strcmp(data_type, "c20") == 0)
printf("\'%s\',\n", c_value[att_array[entry].value_entry]);
    else
```

```
    if (strcmp(data_type, "integer") == 0)
 printf(" %d ,\n", i_value[att_array[entry].value_entry]);
    else
if (strcmp(data_type, "float") == 0)
 printf(" %f ,\n", f_value[att_array[entry].value_entry]);
else
 if (strcmp(data_type, "image") == 0){
  if (image_flag==TRUE)
   image_counter = img_value[att_array[entry].value_entry];
  else
   image_counter = img_record[img_index].i_id;
  printf(" %d ,\n", image_counter);
 }
 else{
  if (sound_flag==TRUE)
   snd_ctr = snd_value[att_array[entry].value_entry];
  else
   snd_ctr = snd_record[snd_index].s_id;
  printf(" %d ,\n", snd_ctr);
 }
   printf("                    ");
   entry = att_array[entry].next_index;
  }
  strcpy(data_type, att_array[entry].data_type);
  if (strcmp(data_type, "c20") == 0)
   printf("\'%s\');\n\n", c_value[att_array[entry].value_entry]);
  else
   if (strcmp(data_type, "integer") == 0)
    printf(" %d );\n\n", i_value[att_array[entry].value_entry]);
   else
    if (strcmp(data_type, "float") == 0)
     printf(" %f );\n\n", f_value[att_array[entry].value_entry]);
    else
     if (strcmp(data_type, "image") == 0){
if (image_flag==TRUE)
 image_counter = img_value[att_array[entry].value_entry];
else
 image_counter = img_record[img_index].i_id;
printf(" %d ,\n", image_counter);

     }
     else{
if (sound_flag==TRUE)
 snd_ctr = snd_value[att_array[entry].value_entry];
else
 snd_ctr = snd_record[snd_index].s_id;
```

```
printf(" %d ,\n", snd_ctr);

     }

/************INSERT STD TUPLE IN INGRES START HERE******************/
/******* **THE INGRES FUNCTION CALLS WRITE MANUALLY****************/
  entry = table_array[table_list[table_cursor]].att_entry;
  count = table_array[table_list[table_cursor]].att_count;
/* # line 2213 "db.sc" */   /* insert */
  {
    printf("\nINSERTING STD TUPLE NOW. PLEASE WAIT!!\n");
    IIsqInit(&sqlca);
    IIwritedb("append to ");
    IIwritedb(table_array[table_list[table_cursor]].table_name);
    IIwritedb("(");
    for (i = 1; i < count; i++)
     {
      IIwritedb(att_array[entry].att_name);
      IIwritedb("=");
      strcpy(data_type, att_array[entry].data_type);
      if (strcmp(data_type, "c20") == 0)
IIsetdom(1,32,0, c_value[att_array[entry].value_entry]);
      else
       if (strcmp(data_type, "integer") == 0)
 IIsetdom(1,30,4, &i_value[att_array[entry].value_entry]);
       else
if (strcmp(data_type, "float") == 0)
 IIsetdom(1,31,4, &f_value[att_array[entry].value_entry]);
else
 if (strcmp(data_type, "image") == 0)
/*  IIsetdom(1,30,4, &img_value[att_array[entry].value_entry]);*/
  IIsetdom(1,30,4, &image_counter);

 else
/*   IIsetdom(1,30,4, &snd_value[att_array[entry].value_entry]);*/
   IIsetdom(1,30,4, &snd_ctr);

      IIwritedb(" ,");
      entry = att_array[entry].next_index;
     }
    IIwritedb(att_array[entry].att_name);
    IIwritedb("=");
    strcpy(data_type, att_array[entry].data_type);
    if (strcmp(data_type, "c20") == 0)
      IIsetdom(1,32,0, c_value[att_array[entry].value_entry]);
    else
```

```
   if (strcmp(data_type, "integer") == 0)
     IIsetdom(1,30,4, &i_value[att_array[entry].value_entry]);
   else
     if (strcmp(data_type, "float") == 0)
IIsetdom(1,31,4, &f_value[att_array[entry].value_entry]);
     else
      if (strcmp(data_type, "image") == 0)
/* IIsetdom(1,30,4, &img_value[att_array[entry].value_entry]);*/
 IIsetdom(1,30,4, &image_counter);
      else
/* IIsetdom(1,30,4, &snd_value[att_array[entry].value_entry]);*/
 IIsetdom(1,30,4, &snd_ctr);
   IIwritedb(" )");
   IIsqSync(3,&sqlca);
   printf("\nINSERT A STD TUPLE COMPLETE!!\n");
  }
/* # line 2261 "db.sc" */   /* insert */
/**************INSERT STD TUPLE IN INGRES STOP HERE*******************/
  printf("\npress ENTER to continue");
  while ((c = getchar()) != '\n')
  ;
  if ((sound_flag == TRUE) || (image_flag == TRUE)){
   if (act_media_count >= 1)
     mod_ql_insert_media_tuple(mode, image_counter, snd_ctr);
   }
 } /* End of ql_insert_tuple() */
```

APPENDIX C: RUNNING THE DATABASE

The system is configured on the SUN server "VIRGO" under the account /n/virgo/work/mdbms/MDBMS/dbrose. The user can set up the path from any account to reach the directory. The files must be copied from the directory to the directory the user will use. When the directory is copied, then the user must log off and log on again so the system may follow the new paths arranged now for proper execution. The database main program executable object code is called *db*. The user must type "db" in the working directory to run the MDBMS prototype. However, a check must be made to ensure that the user is an authorized user to the INGRES DBMS. If the user is not an authorized user, then an error message will appear on the screen stating it is unable to access those areas. If questions arise, then the system administrator should be consulted to acquire a path for accessing the INGRES system.

Db is created using the *make* command, as several other files are needed to run the MDBMS prototype. Make is a command generator that executes a list of shell commands to bring a set of files up to date. Make aligns the dependencies among files. The executable code files are generated by linking the libraries, object files and from the programming language source files. The original program source code is called db.sc. The program is precompiled in the INGRES SQL. This step produces the program in an acceptable C compiler format program db.c. The C compiler creates the object code db.o and links it to the libraries object code listed in the Makefile. Some libraries are Sunwindow library, Sunpixrects library, Suntools library, and INGRES library. More important files that are applications necessary for the prototype are *diction.add*, *imagei_image_facts* and the *prolog_parser*. To check for proper compiling and linking, the user must type *make db* at the prompt. The make command then calls the make file designated for the db executable programming code. The makefile for this db code is not the only copy of a makefile for the MDBMS on the UNIX system. When we worked on our separate modules of the MDBMS, I used a version of the database entitled *dbrose*. For compiling and accessing the dbrose code, the user must type "*make -f Makerose dbrose*" at the UNIX shell prompt in order to ensure that the prototype code is up to date. If the dbrose.sc code needs linking and compiling, then the series of make commands will produce the executable version (dbrose). If db or dbrose are current, a message stating that "db(rose) is up to date" appears on the screen. As one or more of the source files are modified, then recompilation followed by relinking must occur. As in most software engineering projects this process is repeated as each revision is made on the prototype code. An option of make to automatically perform the updating tasks is *$make db*. After this is issued, the Make utility executes the tasks that must compile and relink after a modification.

A *description file* usually defines the dependencies between the files and modules. This description file is usually given the name Makefile. Numerous entries comprise the description file. Each of these entries is delineated by a line that contains a colon. The colon indicates the dependency line and the targets. The targets are on the left side of the colon. The files that the targets depend are on the right of the colon. An example is the following:

```
OBJMODS= ISfunctions.o, comcprologl.o ISroutine.o
#ING_HOME = /ingres
db: db.o $(OBJMODS)
    cc db.c -o db
    /ingres/lib/libqlib/ingres/lib/compatlib
    /-lsuntool -lsunwindow -lpixrect -lm
db.c: db.sc
    esqlc db.sc
```

These examples display the dependency of the file db.c on the file db.sc. The program db.sc is the precompiled by the INGRES SQL.Makefile is shown in Appendix D

APPENDIX D: MAKEFILE AND SAMPLE DICTIONARY

```
MDBMS_PATH = /n/virgo/work/mdbms/MDBMS

PLPATH = /n/virgo/work/mdbms/MDBMS/PROLOG_SOURCE

OBJMODS = ISfunctions.o ISsubroutine.o rpc_pl_call.o plcall_xdr.o\
 plcall_clnt.o CatalogManagement.o SoundModule.o UserInterface.o CreateModule.o \
InsertModule.o Retrieve.o ImageModule.o ModifyModule.o

PLMODS = $(PLPATH)/dict.pl \
 $(PLPATH)/diction.pl \ $(PLPATH)/interface.pl \
 $(PLPATH)/simple.pl \
 $(PLPATH)/list_util.pl \
 $(PLPATH)/read_capt.pl \
 $(PLPATH)/variable.pl \
 $(PLPATH)/gen_util.pl \
 $(PLPATH)/number.pl \
 $(PLPATH)/semantics.pl

DEFINE = defines.h errors.h
Global = GlobalVariables.h
RPC = plcall.h
FLAGS = -g
SERVER = ai9
RSH = rsh
LINT = lint

FILES = Makefile \
rpc_pl_server.c \
rpc_pl_call.c \
plcall.h \
plcall_svc.c \
plcall_xdr.c \
plcall_clnt.c \
IMPORTANT_FILES \
defines.h \
errors.h \
ISsubroutine.c \
ISfunctions.c \
comcprolog_neu.c \

.c.o:; cc -c $(FLAGS) -o $@ $*.c

Retrieve.o CreateModule.o InsertModule.o CatalogManagement.o UserInterface.o \
SoundModule.o ImageModule.o ModifyModule.o: $(Global)
rpc_pl_call.o rpc_pl_server plcall_svc.o plcall_xdr.o plcall_clnt.o: $(RPC)
```

```
Retrieve.o CreateModule.o InsertModule.o CatalogManagement.o UserInterface.o
SoundModule.o\ ISfunctions.o ISsubroutine.o rpc_pl_call.o rpc_pl_server
ImageModule.o ModifyModule.o: $(DEFINE)

db: db.o $(OBJMODS)
@echo "creating DATABASE ..."
cc $(FLAGS) db.o \
$(OBJMODS) \
/ingres/lib/libqlib /ingres/lib/compatlib \
-lsuntool -lsunwindow -lpixrect -lm \
-o db
db.c: db.sc
esqlc db.sc

plcall_xdr_sun4.o: plcall_xdr.c
$(RSH) $(SERVER) cc -c $(FLAGS) \
-o $(MDBMS_PATH)/plcall_xdr_sun4.o \
$(MDBMS_PATH)/plcall_xdr.c

plcall_svc_sun4.o: plcall_svc.c
$(RSH) $(SERVER) cc -c $(FLAGS) \
-o $(MDBMS_PATH)/plcall_svc_sun4.o \
$(MDBMS_PATH)/plcall_svc.c
rpc_pl_server: rpc_pl_server.c plcall_svc_sun4.o plcall_xdr_sun4.o comcprolog_neu.c

$(DEFINE)@echo "creating rpc_pl_server ..."

$(RSH) $(SERVER) cc $(FLAGS) $(MDBMS_PATH)/rpc_pl_server.c \
$(MDBMS_PATH)/plcall_svc_sun4.o \
$(MDBMS_PATH)/plcall_xdr_sun4.o \
-o $(MDBMS_PATH)/rpc_pl_server


prolog_parser: $(PLMODS) $(PLPATH)/diction.add
@echo "creating prolog_parser ..."
sort $(PLPATH)/diction.body $(PLPATH)/diction.add -o $(PLPATH)/diction
cat $(PLPATH)/diction.head $(PLPATH)/diction > $(PLPATH)/diction.pl
rm $(PLPATH)/diction.qof
$(RSH) $(SERVER) qpc -c $(PLPATH)/diction.pl
$(RSH) $(SERVER) qpc -D $(PLPATH)/interface -o $(PLPATH)/prolog_parser
mv $(MDBMS_PATH)/prolog_parser $(MDBMS_PATH)/prolog_parser.lastVersion
cp $(PLPATH)/prolog_parser $(MDBMS_PATH)/prolog_parser

lnt:*.c
$(LINT) $?
@touch lnt


print: $(FILES)
@echo "Print the following files:"
@ls $?
@echo "Interrupt with Control c"
@sleep 3
pr $? | print
@touch
```

SAMPLE DICTIONARY FOR PROLOG PARSER

```
propnoun('Hunter-Liggett',[name('Hunter-Liggett'),place]).
propnoun('Jolon',[name('Jolon'),place]).
propnoun('Nacimiento',[name('Nacimiento'),place]).
propnoun('Pacific',[name('Pacific Ocean'),place]).
propnoun(['Pacific','Ocean'],[name('Pacific Ocean'),place]).
propnoun(['World','War','II'],[name('World War II'),war(ww2)]).
propnoun('Macy''s',[name('Macy''s'),place]).

month('January',[name('January'),month_name]).
month('February',[name('February'),month_name]).
month('March',[name('March'),month_name]).
month('April',[name('April'),month_name]).
month('May',[name('May'),month_name]).
month('June',[name('June'),month_name]).
month('July',[name('July'),month_name]).
month('August',[name('August'),month_name]).
month('September',[name('September'),month_name]).
month('October',[name('October'),month_name]).
month('November',[name('November'),month_name]).
month('December',[name('December'),month_name]).

propnoun('Christmas',[name('Christmas'),holiday_name]).
propnoun('Columbus Day',[name('Columbus Day'),holiday_name]).
propnoun('Independence Day',[name('Independence Day'),holiday_name])./* bh -> nosc alv_images */

noun(atlantic,ML):- noun(ocean, MLtype), union([name(atlantic_ocean),place],MLtype, ML).
noun(destroyer,ML):- noun(warship, MLsup), union([ size(-)], MLsup, ML).
noun(fregate,[ship,fregate]). noun(boat,[ship]).
noun(u-boat,[ship,under_surface_vehicle]).
noun(submarine,[ship,under_surface_vehicle]).
noun(battleship,ML):- noun(warship, MLsup), union([battleship,size(+)], MLsup, ML).
noun(cruiser,ML):- noun(warship, MLsup), union([cruiser, size(-)],MLsup, ML).
noun(gunboat,[ship,gunboat]). noun(freighter,[ship,freighter]).
noun(aircraft_carrier,[ship,aircraft_carrier]).
noun(tanker,[ship,tanker]). noun(cutter,[ship,cutter]).
noun(warship,ML):-noun(ship, MLsup), union([warship,color(gray), hullnumber(number)], MLsup, ML1),
noun(navy,ML2), union(ML1,ML2,ML).
noun(ship,[ship,vehicle]). noun(vessel,[ship,vehicle]).
noun(sea,[sea,geofeature]).
noun(ocean,[sea,geofeature]).
noun(wave,[sea,geofeature]).
noun(lake,[sea(-)]).
noun(pond,[lake(-)]).
noun(navy,[navy,organization]).
/* --- end --- */
noun(road,[road(+)]).
noun(route,[road(+)]).
noun(track,[road(-)]).
noun(path,[road(-)]).
noun(army,[army,organization]).
noun(tank,[tank,vehicle]).
```

```
noun(carrier,[carrier,vehicle]).
noun(jeep,[jeep,vehicle]).
noun(hill,[hill,geofeature]).
noun(mountain,[mountain,geofeature]).
noun(river,[river,geofeature]).
noun(stream,[river,geofeature]).
noun(creek,[river,geofeature]).
noun(arroyo,[river,geofeature]).
noun(junction,[junction]).
noun(intersection,[junction]). /* KMW */
noun(crossroads,[junction]).
noun(plant,[plant]).
noun(vegetation,[vegetation]).
noun(tree,[plant(+)]).
noun(bush,[plant(0)])./* bh */
noun(shrub,[plant(0)]).
noun(brush,[vegetation(-)]).
noun(forest,[vegetation(+)]).
noun(region,[region]).
noun(object,[region]).
noun(area,[region]).
noun(stretch,[region]).
noun(spot,[region]).
noun(shape,[region]).
noun(strip,[strip,region,shape(narrow)]).
noun(side,[side,region]).
noun(half,[half,region,size(big)]).
noun(boundary,[edge,boundary]).
noun(edge,[edge,boundary]).
noun(right,[region,xcoordinate(+)]).
noun(left,[region,xcoordinate(-)]).
noun(middle,[region,xcoordinate(0),ycoordinate(0)]).
noun(north,[region,upper_region,ycoordinate(+)]).
noun(east,[region,right_region,xcoordinate(+)]).
noun(south,[region,lower_region,ycoordinate(-)]).
noun(west,[region,left_region,xcoordinate(-)]).

noun(northeast,[region,upper_right_region,xcoordinate(+),ycoordinate(+)]).
noun(southeast,[region,lower_right_region,xcoordinate(+),ycoordinate(-)]).
noun(southwest,[region,lower_left_region,xcoordinate(-),ycoordinate(-)]).
noun(northwest,[region,upper_left_region,xcoordinate(-),ycoordinate(+)]).
noun(land,[terrain,earth]).
noun(earth,[terrain,earth]).
noun(rock,[terrain,rock]).
noun(grass,[terrain,grass]).
noun(sand,[terrain,sand]). /* KMW */
noun(cover,[terrain]).
noun(terrain,[terrain]).
noun(set,[set]).
noun(group,[set]).
noun(number,[set]).
noun(couple,[set]). /* KMW */
noun(bunch,[set])./* bh */
```

```
noun(convoy,[set]).
noun(fleet,[set]).
noun(motion,[motion]).
noun(morning,[time_loc]).
noun(evening,[time_loc]).
noun(noon,[time_loc]).
noun(midnight,[time_loc]).
noun(afternoon,[time_loc]).
noun(night,[time_loc]).
noun(dusk,[time_loc]).
noun(dawn,[time_loc]).
noun(sunrise,[time_loc]).
noun(sunset,[time_loc]).
noun(second,[time_loc]).
noun(minute,[time_loc]).
noun(hour,[time_loc]).
noun(day,[time_loc]).
noun(week,[time_loc]).
noun(month,[time_loc]).
noun(year,[time_loc]).
noun(decade,[time_loc]).
noun(past,[time_loc]).
noun(present,[time_loc]).
noun(future,[time_loc]).
noun(spring,[time_loc]).
noun(summer,[time_loc]).
noun(autumn,[time_loc]).
noun(fall,[time_loc]).
noun(winter,[time_loc])
. /* --- end ---*/
noun(turn,[turn]).
noun(bend,[turn]).
noun(line,[line]). /* KMW */

adjective(white,[color(white)]).
adjective(black,[color(black)]).
adjective(gray,[color(gray)]).
adjective(rough,[texture(rough)]).
adjective(mixed,[texture(rough)]).
adjective(heterogeneous,[texture(rough)]).
adjective(smooth,[texture(smooth)]).
adjective(homogeneous,[texture(smooth)]).
adjective(forested,[terrain(forested)]).
adjective(bare,[terrain(unforested)]).
adjective(scattered,[dispersion(wide)]).
adjective(scarce,[dispersion(wide)]).
adjective(clustered,[dispersion(narrow)]).
adjective(big,[size(+)]). adjective(large,[size(+)]).
adjective(small,[size(-)]). adjective(tiny,[size(-)]).
adjective(long,[size(+)])./* bh */
adjective(tall,[height(+)]).
adjective(short,[height(-)]).
adjective(flat,[height(-)]). /* KMW */
```

```
adjective('American',[nationality(us)]).
adjective('US',[nationality(us)]).
adjective('Russian',[nationality(ussr)]).
adjective('Soviet',[nationality(ussr)]).
adjective('NATO',[nationality(nato)]).
adjective(american,[nationality(us)]).
adjective(us,[nationality(us)]).
adjective(russian,[nationality(ussr)]).
adjective(soviet,[nationality(ussr)]). /* bh */
adjective(german,[nationality(germany)]).
adjective(british,[nationality(uk)]).
adjective(high,[height(+)]).
/* --- end --- */
adjective(nato,[nationality(nato)]).
adjective(upper,[ycoordinate(+)]).
adjective(lower,[ycoordinate(-)]).
adjective(right,[xcoordinate(+)]).
adjective(left,[xcoordinate(-)]).
adjective(middle,[xcoordinate(0),ycoordinate(0)]).
adjective(broad,[width(+)]).
adjective(wide,[width(+)]).
adjective(narrow,[width(-)]).
adjective(slim,[width(-)]).
adjective(other,[cardinality(1),different]).
adjective(single,[cardinality(1)]).
adjective(first,[cardinality(1)]).
adjective(lone,[cardinality(1)]).
adjective(one,[cardinality(1)]).
adjective(second,[cardinality(1),different]).
adjective(separate,[cardinality(1),different]).
adjective(two,[cardinality(2)]).
adjective(three,[cardinality(3)]).
adjective(few,[cardinality(-)]).
adjective(many,[cardinality(+)]).
adjective(various,[cardinality(+)]). /* KMW */
adjective(north,[direction(0)]).
adjective(east,[direction(90)]).
adjective(south,[direction(180)]).
adjective(west,[direction(270)]).
adjective(northeast,[direction(45)]).
adjective(northwest,[direction(315)]).
adjective(southeast,[direction(135)]).
adjective(southwest,[direction(225)]).

determiner(the,[definite]).
determiner(a,[indefinite]).
determiner(an,[indefinite]).
determiner(some,[indefinite]).
determiner(several,[indefinite]).
determiner(one,[definite,cardinality(1)]).
determiner(two,[definite,cardinality(2)]).
determiner(no,[definite,cardinality(0)]).
determiner(many,[definite,cardinality(+)]).
```

```
determiner(another,[definite,cardinality(1),different]).
verb(have,[transitive]).
verb(own,[transitive]).
verb(know,[transitive]).
verb(reach,[transitive]).
verb(go,[transitive]).
verb(head,[transitive]).
verb(leave,[transitive]).
verb(visit,[transitive]).
verb(see,[transitive]).
verb(view,[transitive]).
verb(arrive,[intransitive]).
verb(remain,[intransitive]).
verb(stay,[intransitive]).
verb(depart,[intransitive]).
verb(come,[intransitive]).
verb(terminate,[intransitive]).
verb(end,[intransitive]).
verb(run,[intransitive]).
verb(turn,[intransitive]).
verb(bend,[intransitive]).
verb(cut,[transitive]).
verb(separate,[transitive]).
verb(shape,[transitive]).
verb(cross,[transitive]). /* KMW */
verb(join,[transitive]). /* KMW */
verb(move,[motion,intransitive]). /* bh */
verb(sail,[motion,intransitive]). /* bh */

aux(will,[tense(future)]).
aux(can,[enablement]).
aux(may,[possibility]).
aux(has,[tense(past),singular]).
aux(have,[tense(past),plural]).

adverb(soon,[time(short)]).
adverb(quickly,[time(short)]).
adverb(north,[direction(0)]).
adverb(east,[direction(90)]).
adverb(south,[direction(180)]).
adverb(west,[direction(270)]).
adverb(westward,[direction(270)]). /* bh */
adverb(northeast,[direction(45)]).
adverb(northwest,[direction(315)]).
adverb(southeast,[direction(135)]).
adverb(southwest,[direction(225)]).
adverb(tomorrow,[time(1)]).
adverb(now,[time(0)]).
adverb(today,[time(0)]).
adverb(yesterday,[time(-1)]).
adverb(often,[frequency(high)]).

doesword(does,[tense(present)]).
```

```
doesword(do,[tense(present)]).
doesword(did,[tense(past)]).
doesword(has,[tense(past),singular]).
doesword(have,[tense(past),plural]).

tobe(is,[tense(present)]).
tobe(was,[tense(past)]).
tobe(were,[tense(past),plural]).
tobe(were,[tense(subjunctive),singular]).

preposition(at,[property(location)]).
preposition(at,[property(time_spec)]).
preposition(in,[property(inside)]).
preposition(in,[property(in_period)]).
preposition(within,[property(inside)]).
preposition(within,[property(in_period)]).
preposition(through,[property(part_inside)]).
preposition(between,[property(part_inside)]).
preposition(among,[property(part_inside)]).
preposition(to,[property(destination)]).
preposition(to,[property(time_dest)]).
preposition(from,[property(source)]).
preposition(from,[property(time_src)]).
preposition(for,[property(purpose)]).
preposition(for,[property(beneficiary)]).
preposition(for,[property(time_spec)]).
preposition(with,[property(contains)]).
preposition(with,[property(coagent)]).
preposition(with,[property(tool)]).
preposition(above,[property(above)]).
preposition(on,[property(above)]).
preposition(over,[property(above)]).
preposition(below,[property(below)]).
preposition(along,[property(beside)]).
preposition(beside,[property(beside)]).
preposition(of,[property(subtype)]).
preposition(of,[property(part_of)]).
preposition(of,[property(owner)]).
preposition(before,[property(time_spec)]).
preposition(after,[property(time_spec)]).
preposition(until,[property(time_spec)]).
preposition(during,[property(time_spec)]).

clausehead(that,[]).
clausehead(which,[]).
clausehead(who,[]).

conjunction(and,[conjunction(and)]).
conjunction(or,[conjunction(or)]).
conjunction(but,[conjunction(and)]).
conjunction(plus,[conjunction(and)]).

punctuation(comma,[]).
```

LIST OF REFERENCES

[Ref. 1] Atila, Y.V., *Design and Implementation of a Multimedia DBMS: Sound Management Integration*, Master's Thesis, Naval Postgraduate School, Department of Computer Science, Monterey, CA, December 1990.

[Ref. 2] Aygun, H., *Design and Implementation of a Multimedia DBMS: Complex Query Processing*, Master's Thesis, Naval Postgraduate School, Department of Computer Science, Monterey, CA, September 1991.

[Ref. 3] Bertino, E., Gibbs, S., Rabitti, F., Thanos, C., and Tsichritzis, D., *A Multimedia Document Server*, in Proc. 6th Japanese Advanced Database Symposium (Tokyo, Aug. 1986), Information Processing Society of Japan, 1986, pp.123-134.

[Ref. 4] Bertino, E., Rabitti, F., Gibbs, S., *Query Processing in a Multimedia Document System*, ACM Trans. on Office Information Systems, Vol. 6, no. 1, Jan. 1988, pp.1-41.

[Ref. 5] Chrisodoulakis, S., Theodoridou, M., Ho, F., Papa, M., and Pathria, A., *Multimedia Document Presentation, Information Extraction, and Document Formation in MINOS: A Model and a System,* ACM Transactions on Office Information Systems, vol. 4, no. 4, Oct. 1986, pp. 345-383.

[Ref. 6] Dulle, J., *The Scope of Descriptive Captions for Use in a Multimedia Database System,* Master's Thesis, Naval Postgraduate School, Department of Computer Science, Monterey, CA, June 1990.

[Ref. 7] Kiem, D., and Lum, V. Y., *A Graphical Database Interface for a Multimedia DBMS,* Report no. NPS52-91-013, Naval Postgraduate School, Monterey, CA, June 1991.

[Ref. 8] Kosaka, K., Kajitani, K., and Satoh, M., *An Experimental Mixed-Object Database System,* in Proc IEEE CS Office Automation Symposium (Gaithersburg, MD, April 1987), IEEE CS Press, order no. 770, Washington 1987, pp. 57-66.

[Ref. 9] Lum, V. Y., and Meyer-Wegener, K., *A Conceptual Design of a Multimedia DBMS for Advanced Applications,* Report no. NPS52-88-025, Naval Postgraduate School, Monterey, CA, August 1988.

[Ref. 10] Lum, V. Y., and Meyer-Wegener, K., *A Multimedia Database Management System Supporting Contents Search in Media Data,* Report no. NPS52-89-020, Naval Postgraduate School, Monterey, CA, March 1989. Also in Advances in Computing and Information, Proceedings of the International

Conference on Computing and Information (ICCI '90), Niagra Falls, Canada, May 23-26, 1990 and to appear in Lecture Notes in Computer Science, Springer Verlag.

[Ref. 11] Meyer-Wegener, K., Lum, V. Y., and Wu, C.T., *Image Database management in a Multimedia System, in Visual Database Systems,* (IFIP TC2/G2.6 Working Conference, Tokyo, Japan, April 3-7, 1989), Ed. T. L. Kini;, North-Holland, Amsterdam 1989, pp. 497-523.

[Ref. 12] Meyer-Wegener, K., et al., *Managing Multimedia Data,* Report no. NPS52-88-010, Naval Postgraduate School, Monterey, CA, March 1988.

[Ref. 13] Peabody, C., *Design and Implementation of a Multimedia DBMS: Graphical User Interface Design and Implementation,* Master's Thesis, Naval Postgraduate School, Monterey, CA, September 1991.

[Ref. 14] Pei, S., *Design and Implementation of a Multimedia DBMS: Catalog Management, Table Creation and Data Insertion,* Master's Thesis, Naval Postgraduate School, Monterey, CA, December 1990.

[Ref. 15] Pongsuwan, W., *Design and Implementation of a Multimedia DBMS: Retrieval Management,* Master's Thesis, Naval Postgraduate School, Monterey, CA, December 1990.

[Ref. 16] Rowe, N., and Guglielmo, E., *Exploiting Captions for Access to Multimedia Databases,* to appear in IEEE Computer, 1991.

[Ref. 17] Sawyer, G., *Managing Sound in a Relational Multimedia Database System,* Master's Thesis, Naval Postgraduate School, Department of Computer Science, Monterey, CA, December 1988.

[Ref. 18] Stonebraker, M., and Rowe, L., *The Design of POSTGRES,* Proc. SIGMOD Conference, Washington D. C., May 1986.

[Ref. 19] Thomas, C.A., *A Program Interface Prototype for a Multimedia Database Incorporating Images,* Master's Thesis, Naval Postgraduate School, Department of Computer Science, Monterey, CA, December 1988.

[Ref. 20] Wu, C.T., Nardi P., Turner, H., Antonopoulos, D., *Argos Next Generation Shipboard Information Management System,* Report no. NPS52-90-006, Naval Postgraduate School, Department of Computer Science, Monterey, CA, December 1989.

[Ref. 21] Woelk, D. and Kim, W. *Multimedia Management in an Object-Oriented Database System,* Proc. 13th Int. Conf on VLDB, Brighton (England), September 1987.

INITIAL DISTRIBUTION LIST

Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
Dudley Knox Library Code 052 Naval Postgraduate School Monterey, California 93943-5100	2
Center for Naval Analysis 4401 Ford Ave. Alexandria, Virginia 22302-0268	1
John Maynard Code 42 Command and Control Departments Naval Ocean Systems Center San Diego, California 92152	1
Dr. Sherman Gee ONT-221 Chief of Naval Research 880 N. Quincy Street Arlington, Virginia 22217-5000	1
Leah Wong Code 443 Command and Control Departments Naval Ocean Systems Center San Diego, California 92152	1
Professor Vincent Y. Lum Code CsLm Computer Science Department Naval Postgraduate School Monterey, CA 93943	2
Rosemary E. Stewart 1125 Park Avenue Port Hueneme, California 93041	2

Professor C. Thomas Wu 1
Code CsWu
Department of Computer Science
Naval Postgraduate School
Monterey, California 93943

Dr. Bernhard Holtkamp 1
University of Dortmund
Software Technology
P.O. Box 500
D-4600 Dortmund 50 / GERMANY

Professor Klaus Meyer-Wegener 1
University of Erlangen-Nuernberg
IMMD VI, Martensstr. 3,
5250 Erlangen / GERMANY